Update Sets

Moving Customizations from Instance to Instance

System Update Sets

Note: *This article applies to Fuji and earlier releases. For more current information, see System Update Sets* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.'**

Overview

An *update set* is a group of customizations that can be moved from one instance to another. This feature allows administrators to group a series of changes into a named set and then move them as a unit to other systems. In most cases, update sets allow customizations to be developed in a development instance, moved to a test instance, and then applied to a production instance.

Before using update sets, review the Getting Started with Update Sets page.

Update Sets Process

A common process for developing customizations with update sets involves moving changes from development to test to production instances:

1. Create an update set on the development instance.
2. Make customizations and changes on the development instance.
3. Mark the update set as **Complete**.
4. Log in to the test instance and retrieve the completed update set from the development instance.
5. Commit the update set on the test instance, and test customizations thoroughly.
6. If the update set has problems in the test instance, repeat steps 1 - 5 to develop the fix on the development instance with another update set.
7. Log in to the production instance and retrieve the completed update set from the development instance. If the update set required a fix, retrieve both update sets.
8. Commit the update set on production. If the update set required a fix, commit both update sets in the order they were made.

If your development environment consists of only two instances, you can combine your development and testing instances into a single staging instance. For example:

1. Create an update set on the staging instance.
2. Make customizations and changes on the staging instance.
3. Mark the update set as **Complete**.
4. Test customizations thoroughly on the staging instance.
5. If the update set has problems, repeat steps 1 - 4 to develop the fix on the staging instance with another update set.
6. Log in to the production instance and retrieve the completed update set from the development instance. If the update set required a fix, retrieve both update sets.
7. Commit the update set on production. If the update set required a fix, commit both update sets in the order they were made.

To learn more, see Using Update Sets.

Update Sets Tables

Each update set is stored in the Update Set [sys_update_set] table, and the customizations that are associated with the update set, which are entries in the Customer Update [sys_update_xml] table, appear as a related list on the update set record.

When a tracked object is customized, a corresponding record is added or updated in the Customer Update [sys_update_xml] table and is associated with the user's current update set. The associated application file properties are tracked and transferred along with the customized object in a single update record. A corresponding record is also added to the Versions [sys_update_version] table.

The Customer Update table contains one record per customized object, per update set. The Versions table contains one record per change to a customized object.

- Administrators can compare two versions and revert to a specific version of an object.
- Administrators can suppress versions for specific tables.
- Administrators can specify fields on tracked tables that you can change without skipping updates to the rest of the record.

Note: *Do not directly modify Customer Updates [sys_update_xml] records.*

How the system determines the default update set (Fuji)

Only one update set can be the default set for any scope. To set an update set to be the default set, you set the **Default set** field to true.

When you set **Default set = true**, the following actions occur:

- The update set becomes the default update set for its scope
- The system sets **Default set = false** for all other update sets with the same scope. (This ensures that there is only one default update set for each scope.)

Global default update set

The global default update set is the set where **Default set = true** and application scope is **global**.

The global default update set (regardless of the **Name** of the set) provides system functionality and should not be changed, deleted, or moved between systems. Use this update set to make changes to an instance without adding the changes to any user-created update sets.

When the system auto-generates a default update set

At all times, to ensure that no updates to an instance are lost, the system ensures that there is a default set for the user's scope. If the system finds that a default update set does not exist (or is marked **Ignored** or **Completed**) for the current scope, then the system auto-generates an update set and sets **Default set = true**. Here are some common cases:

Upon first login

The very first time that an admin logs in, the system sets the system's global default update set as the admin's update set. In addition, the application picker sets the admin's Application scope to **global**.

If a global default update set does not exist (or is marked **Ignored** or **Completed**), the system creates a new update set for the global application scope and performs the following actions:

- The system sets **Default set = true** for the new set.
- The system sets the name of the new set to start with the name of the former default set and appends the next numeral (in the sequence SetName, SetName 1, SetName 2, …, SetName n).
- The system sets the newly created set as the admin's update set.

When a user marks the default set for a scope as Completed or Ignored (not a recommended practice)

When a user marks the default set for a scope as **Completed** or **Ignored**, the system immediately auto-generates a new default set for the scope.

Changing application scope

If you change application scope and: Your preferred update set is Complete or Ignored and there is no In-Progress default update set for the new scope, then the system auto-generates a default update set for the scope.

Default update sets (Eureka and earlier)

All update sets are **global** (there is no concept of scope).

The system determines which update set is the default set based on the state and the sort order of the names of the update sets -- the first default update set lexicographically that is in the **In Progress** state is the default set. For example, for an instance with the following update sets, Default 1 is **Completed** and Default 2 is **In Progress**. Therefore, Default2 is the default set:

ASet, BSet, Default, Default 1, Default 2, Default 3, Default 4

Naming non-default sets

For any update set that should not be a default set, do not start the name with the text "Default".

Upon first login

The very first time that an admin logs in, the system sets the system's default update set as the admin's update set. If a default update set does not exist (or is marked **Ignored** or **Completed**), the system creates a new update set and performs the following actions:

- The system sets the name of the new set to start with "Default" and appends the next numeral (in the sequence Default, Default 1, Default 2, …, Default n).
- The system sets the newly created set as the admin's update set.

Determining Which Customizations Are Tracked

Update sets can track customizations to application tables, fields, and records. Update sets track customizations under these conditions:

- Where the table has an `update_synch` dictionary attribute.
- Where there is a special handler to track changes to multiple tables.
- Where the administrator has not specifically excluded a field from updates.

In general, update sets capture configuration information but not task or process data. For example, update sets track service catalog item definitions and related configuration data like variables and variable choices. However, if you test the service catalog by placing orders, the order requests, items, and catalog tasks are not tracked by update sets.

Update sets have a limited capacity to transfer data as application files. This is intended to provide demo data for applications. For larger data transfers export data and import it with an import set or import set web service.

The update_synch Attribute

To see a list of tables on which customizations are tracked, navigate to **System Definition > Dictionary** and filter on **attributes CONTAINS update_synch**.

A default rule blocks the use of the `update_synch` attribute on a table for which it is not predefined to avoid the following issues:

- Some core tables require special update handling because they represent information on multiple tables. When the `update_synch` attribute is added to these tables, duplicate update records are created, causing major conflicts that are difficult to troubleshoot and repair.
- Using the `update_synch` attribute to migrate data records between instances can cause performance issues, because it is not intended for this purpose. To migrate data, use an instance-to-instance import.

⚠	**Warning:** Do not add the `update_synch` attribute to a dictionary record. When improperly used, this attribute can cause major performance issues or cause the instance to become unavailable. Adding this attribute is **not supported**.

Special Handlers

Some changes require special handlers because they represent information on multiple tables. These changes are packaged into one update set entry so that all records are properly updated when the customization is committed. The following changes are tracked with special handlers:

- Choice lists
- Dictionary entries
- Field labels
- Form sections
- Lists
- Related lists
- Workflows

Choice Lists

Update sets store both new and updated choice options as separate records in the Update Version [sys_update_version] and Customer Update [sys_update_xml] tables (starting with the Eureka release). For example, suppose you create a new Activity [u_activity] table that extends the Task table and add a new choice option to the Task state field that is only visible for your extended table (for example, **My State**). When you publish these changes as an update set, the update only contains update and version records for the choice you added to the u_activity table. The choice options in the task table are unaffected. In addition, you have the option to move the u_activity table and its associated choice to a separate application without affecting the default ServiceNow task application.

In versions prior to Eureka, choice option updates were collected into a single update/version record. For example, suppose you create a new Activity [u_activity] table that extends the Task table and add a new choice option to the Task state field that is only visible for your extended table (for example, **My State**). When you publish these changes as an update set, the update contains all existing choice options from the task table as well as choices from the u_activity table.

Dictionary Changes

Update sets prevent you from applying changes that result in data loss such as:

- Removing tables
- Changing a column's data type

Update sets do not track the removal of tables from the system dictionary. Instead, customers must manually remove tables from the target instance.

While update sets track data type changes, the target instance skips any change that results in data loss and instead adds a log message about the action. Customers can use the log to manually make data type changes on the target instance.

Note: *Update set previews no longer check for type mismatch problems since the target instance skips changes resulting in data loss (starting with the Dublin release).*

Homepages and Content Pages

Homepages, content pages, and Performance Analytics dashboards are not added to update sets by default. You must manually add pages to the current update set by unloading them. See:

- Adding Homepages to Update Sets
- Adding Content Pages to Update Sets
- Adding Performance Analytics Dashboards to Update Sets

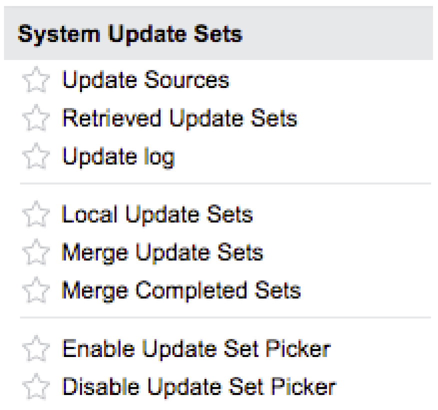
Application Changes

The system creates a separate update set for each application that only contains changes associated with the application. This ensures that each application's access settings are properly evaluated and applied when committing update set changes.

This feature is available starting with the Fuji release.

Menus and Modules

To access update sets features, use these modules under the **System Update Sets** menu.



System Update Sets application

- **Update Sources:** defines the instances to retrieve update sets from. See Transferring Update Sets.
- **Retrieved Update Sets:** lists the update sets that have been retrieved from update sources.
- **Update Log:** shows the history of committing update sets.
- **Local Update Sets:** lists the update sets created on the instance. See Creating Update Sets.
- **Merge Update Sets:** provides the ability to merge update sets. By default, the list is filtered to show only update sets that are in progress.
- **Merge Completed Sets:** provides the ability to merge update sets (starting with the Dublin release). By default, the list is filtered to show only completed update sets.
- **Enable Update Set Picker:** enables the Update Set picker in the banner frame.
- **Disable Update Set Picker:** disables the Update Set picker.

Enhancements

Fuji

- Tracks each application in a separate update set.
- Tracks workflows with a single update record that only contains the latest version of the workflow as a **Workflow** update type.

Eureka

- Requires remote instances be on the same major version as the local instance to retrieve update sets.
- Allows update sets to store choice list options as changes to an extended table rather than changes to a parent table.

Dublin

- The Update Source table label is changed to Remote Instance, and the table is used both to retrieve remote update sets and to define relationships between instances in team development.
- You can test the connection when defining a remote instance. If a connection fails, a warning message identifies the reason the connection was unsuccessful. You must establish connectivity before you can save the connection settings.
- When you retrieve update sets from a remote instance, the confirmation page provides messages about how many update sets were transferred and which were ignored.

- You cannot edit remote update set records.
- When you preview update sets, the completion page provides messages about how many problems are detected. Collisions are detected as problems. The default proposed action for resolving any problem is **Commit**.
- Before you commit an update set, you must address all problems. When you have chosen to either accept or skip each problematic update, the **Commit Update Set** action becomes available.
- You can preview update sets that have been backed out. You cannot preview update sets that have been committed.
- Update records corresponding to dictionary entries can apply database changes that do not result in data loss. An update record with a type of **Database field(s)** is no longer required.
- When you create a table, update records are created for the table record on the Table [sys_db_object] table and the collection record on the Dictionary Entry [sys_dictionary] table.
- You can revert database changes.
- You can merge update sets in any state, including completed update sets. You can use filters to find the update sets you want to merge.
- In the Versions [sys_update_version] table, the **State** field shows whether version is or has ever been loaded on the instance. For the current version of a record, the field is highlighted in green.

References

[1] https://docs.servicenow.com/bundle/jakarta-application-development/page/build/system-update-sets/concept/system-update-sets.html

Getting Started

Note: *This article applies to Fuji and earlier releases. For more current information, see Get Started with Update Sets* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

Because update sets make changes to an instance, review the following best practice information to avoid errors and performance issues. Learn how to plan the update process and avoid common pitfalls.

When to Use Update Sets and When to Use Another Deployment Option

Deployment option	Good for	Future considerations
Update Sets	• Storing changes to a baseline or installed application. • Storing and applying a particular version of an application. • Producing a file for export.	• You can manually create update sets to store a particular application version. • Use update sets to deploy patches or changes to installed applications. **Note:** *Avoid using update sets to install applications. Instead, use the application repository or the ServiceNow Store to install applications.*
Application Repository	• Installing and updating applications on all of your instances • Automatically managing application update sets • Restricting access to applications to the same company • Deploying completed applications to end users	• Consider uploading an application to the ServiceNow Store to share it with other users. • Allows installation of and update to the latest application version only. Use update sets to store prior application versions. **Note:** *If used in conjunction with team development, make applications available only from a parent instance.*
Team Development	• Providing change management across multiple instances • Allowing multiple developers to work on applications • Organizations that have access to several sub-production instances	• Works best when each development team has access to a dedicated development instance. • Requires developers to manually merge colliding changes. • Works only for instances owned by the same organization. **Note:** *If used in conjunction with the application repository, make applications available from a parent instance.*

Planning the Update Process

Before working with update sets, create a standard process for moving customizations from instance to instance. Review the following items to ensure that there are no problems during the update set process:

1. Check that both instances are the same version.

   Customizations may not work if they rely on code that has changed between versions.
2. Determine the changes to make in a single update set.

   ServiceNow recommends that you complete your update sets as you finish small to medium-sized tasks. As update sets get larger, it becomes harder to review them, takes longer to identify specific changes within them, increases the risk of conflicts with other update sets, and takes more time to preview and commit them. This is especially true if the update sets contain schema changes or revisions to large workflows or if the set has to be backed out.
3. Ensure that all base system records have matching **sys_id** fields.

   Some base system records are created on an instance after provisioning and do not match between different instances, leading to problems with update sets. The best way to avoid this issue is to:

   1. Provision production and sub-production instances.
   2. Clone the production instance onto the sub-production instance.
4. Identify a common path for update sets to move from instance to instance and maintain that model.

   Never migrate the same update set from multiple sources. Best practice is to move update sets from dev to test and then from test to production.
5. Plan for when to commit the update set to production.

Avoid committing an update set to a production instance during business hours. The instance may perform slower temporarily as the update set applies.

6. Make sure update set names are clear.

   Create a naming convention to coordinate changes from multiple developers and to reference when committing the changes to another instance.

   - If update sets are being generated as fixes for problems, consider including the problem ticket in the name (for example, **PR10005 - Duplicate Email Issues Fix**).
   - If more than one update set is needed to address a problem, include a sequence number in the naming convention so that update sets are applied in the order that they were created (for example, **PR10005 - Duplicate Email Issues Fix** and **PR10005.2 - Duplicate Email Issues Fix**).
7. Know how update sets work.
   - What records are generated for an update set
   - Which customizations are tracked by update sets
   - Which dictionary changes are valid for update sets
   - Which customizations can be reversed once applied
8. Before making any customizations, double-check that the correct update set is selected.

Avoiding Common Pitfalls

In addition to planning the process, make sure to avoid common pitfalls:

1. Do not delete update sets.

   If an update set is deleted, any updated records may be overwritten in the next update.
2. Do not include the **system_id** field from the ldap_server_config record in an update set.

   An update set from a working configuration points to the wrong system_id node for the target instance and does not work.
3. Do not back out the Default update set. This action causes damage to the system.
4. Never change the update set field value (*update_set*) in a Customer Update record (*sys_update_xml*).

   If a customization is made in the wrong update set, take the following action:

   1. Switch to the desired update set.
   2. Modify the object (record) that was originally changed. You can make a trivial change, such as adding a field.
   3. Save the record.
   4. Back out the change just performed, and then save the record again.

   This action ensures that the latest version of the object is included in the desired update set and prevents duplicate updates for the same object in a single update set.
5. Do not mark an update set as **Complete** until it is ready to migrate.

   Once an update set is complete, do not change it back to **In progress**. Instead, create another update set for the rest of the changes, and make sure to commit them together in the order that they were created. Naming conventions may help in this case (for example, **Performance Enhancements** and **Performance Enhancements 2**).
6. Do not manually merge updates into an update set.

   Always use the **Merge Update Sets** module. This tool compares duplicate files between update sets and selects the newest version.
7. If a committed update set has a problem in the test instance, build the fix in another update set in the development instance. Commit this set to the test instance, and then make sure both sets are migrated to the production instance and committed in the order they were made.

8. Always preview an update set before committing it.
9. Set completed update sets on the production instance to **ignore**.

   This state ensures the update set is not reapplied when cloning the instance.
10. Keep a to-do list of manual changes and data loads that need to be completed after an update set is applied.
11. Do not make too many changes at one time.

    Verify that the correct changes have been made incrementally.

References

[1] https://docs.servicenow.com/bundle/jakarta-application-development/page/build/system-update-sets/reference/get-started-update-sets.html

Administering Update Sets

Note: *This article applies to Fuji. For more current information, see Update Set Administration* [1] *at* http://docs.servicenow.com
The Wiki page is no longer being updated. Please refer to http://docs.servicenow.com for the latest product documentation.

Overview

Administrators can configure options for update sets, such as excluding certain fields from updates and controlling access to the Update Set picker.

Excluding Fields from Updates

Administrators can specify fields on tracked tables that you can change without skipping updates to the rest of the record. This feature applies to changes you make after the Calgary release. During subsequent software upgrades, the value of the excluded field is preserved, while the rest of the record receives updates. For example, you may want to select the **Client callable** check box for a script include, but still receive upgrades to the script.

To exclude a field from updates, add this dictionary attribute to the field:

```
update_exempt
```

Note:

- Values for excluded fields are not retained when you revert customizations to a default software version. For example, you activate a UI macro and change the XML script. Later, a software upgrade contains a feature for the macro that you would like to implement, so you revert your customizations. The default version replaces the entire customized version, and you now need to reactivate the macro.
- If you also change a field that is not update_exempt, then updates are skipped for the entire record (the entire customization is preserved during upgrades).

Tracking the Active Field

By default, the **Active** field on a tracked table is treated as update_exempt even if the attribute is not present (starting with the Calgary release). Any changes that are only the result of changing the **Active** field are update_exempt as well. This allows you to change the field value without affecting the **Updated** and **Updated By** system fields.

To specify that changing the **Active** field preserves the entire record (it is not excluded), add the following attribute to the **Active** field on the table:

```
update_exempt=false
```

Overwriting Customizations During Upgrades

When you change any non-excluded fields on a record, a corresponding record is added in the Customer Update [sys_update_xml] table and the **Replace on upgrade** field is set to **false**. To prevent customizations from being overwritten by system upgrades, the upgrade process automatically skips changes to these objects.

You may want to overwrite your customizations with the next software version. For example, you may change a script to implement a temporary workaround for a problem that is fixed in the next version. You would want to overwrite your workaround when upgrading to the next version to ensure that you receive any future enhancements to the script.

1. Open the customized object (for example, the *ArrayUtil* script include).
2. Right-click the header and select **Show Latest Update**.

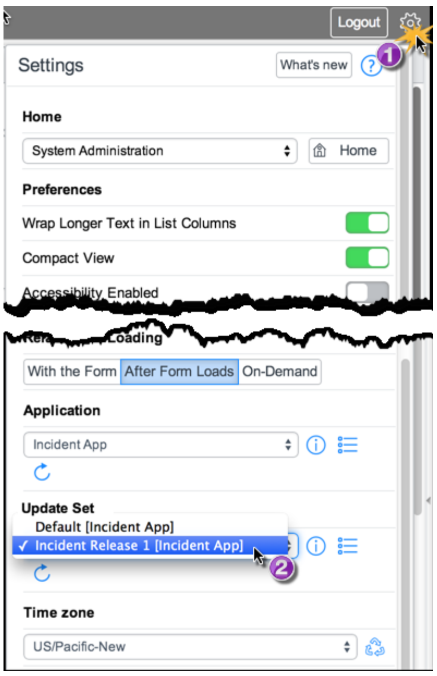
   Alternatively, right-click the header and select **Show Application File** (starting with the Dublin release).
3. Configure the form to add the **Replace on upgrade** field, if necessary.
4. Select the **Replace on upgrade** check box and click **Update**. The customized object will be replaced on the next upgrade.

Granting Access to the Update Set Picker

The Update Set picker appears on the Settings panel. The picker allows users to choose an update set for tracking customizations. By default, only administrators can use the the Update Set picker. To grant access to additional users:

1. Add the following system property.
   - **Name:** *glide.ui.update_set_picker.role*
   - **Value:** enter the role that has access to the Update Set picker (in addition to administrators).

   Note: This property accepts only one role. To give access to multiple roles, create a new role (example, update_set_picker) and include that role in the roles that need access.
2. Click **Submit**.



Update Set picker

Note: *Users must have permission to read the Update Sets [sys_update_set] table to see any choices in the Update Set picker.*

To remove the Update Set picker, navigate to **System Update Sets > Disable Update Set Picker.** To restore it, navigate to **System Update Sets > Enable Update Set Picker**.

References

[1] https://docs.servicenow.com/bundle/jakarta-application-development/page/build/system-update-sets/reference/update-set-administration.html

Using Update Sets

Note: *This article applies to Fuji and earlier releases. For more current information, see Update Set Use* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.**

Overview

An update set is a group of customizations that can be moved from one instance to another. For example, a set of enhancements to incident management can be grouped in an update set called **Incident Management 2.0**. While **Incident Management 2.0** is marked as the current update set, all changes are tracked in it.

Before using update sets, review the Getting Started with Update Sets page to learn when to use and when not to use update sets, and how to plan the update process and avoid common pitfalls. Then, create an update set and use it to make changes on a development instance. You can report on updates, merge update sets, and compare update sets to ensure the desired changes are ready to move.

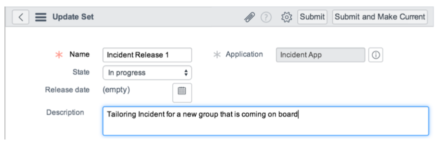
When the update set is completed, you can transfer the update set to another instance according to your test process. See Transferring Update Sets.

Note: *Properties that are tagged as Private are excluded from update sets. Keeping system properties private prevents settings in one instance from overwriting values in another instance. For example, you may not want a system property in a production instance to use a particular value from a development instance. See Adding a Property.*

Creating Update Sets

To create a new update set:

1. Navigate to **System Update Sets > Local Update Sets** and click **New**.
2. Enter update set details (see table). For more information, see Getting Started with Update Sets and Update Sets Best Practices.
3. Click **Submit** to create the update set. Starting with the Fuji release: if the picker is enabled and the update set is in the **In progress** state, click **Submit and Make Current** to:
   - Create the update set
   - Select the new update set in the Update Set picker
   - Add customizations to the new update set.



Name: Incident Release 1
Application: Incident App
State: In progress
Release date: (empty)
Description: Tailoring Incident for a new group that is coming on board

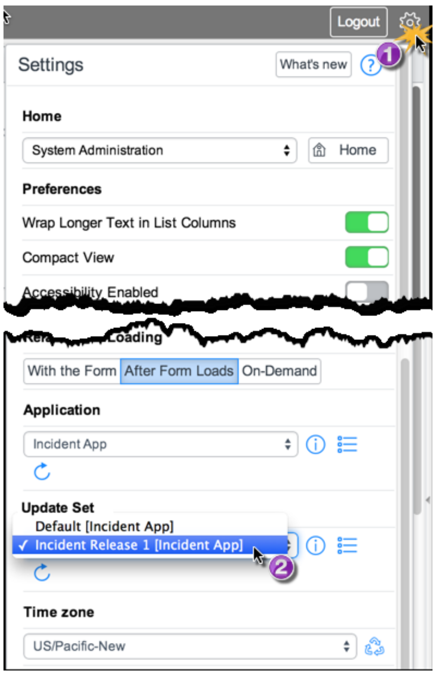
Update set record

Field	Description
Name	Enter a unique name for the update set. You can use naming conventions to organize update sets. For example, add the problem number to the name of the update that fixes it, identify the application scope, or use sequence numbers to keep track of the order in which update sets need to be committed.
State	Select **In progress** for a new update set. Selecting an **In progress** update set tracks customizations in the update set. The Update Set picker only displays **In progress** update sets. Select **Completed** only when you are certain that the update set is complete. After it is marked **Completed**, do not set it back to **In progress**. Instead, create a new update set with further customizations, and make sure to commit the update sets in the order that they were marked **Completed**. Use **Completed** update sets to transfer changes from one instance to another. Select **Ignore** when you are no longer working on an update set but do not want it to be transferred to another instance. You should always set **Completed** update sets on the production instance to **Ignore**. This state ensures the update set is not committed again when cloning the instance.
Created By	Populates your user name when you submit a new update set record.
Created	Populates the timestamp when you submit a new update set record.
Release Date	Enter the date on which you plan to release the update set.
Application (starting with the Fuji release)	Populates the application scope that is currently selected in the Application picker. All changes in the update set apply only to the current scope.
My Current Set (prior to the Fuji release)	Select the check box to add your customizations to the current update set. Selecting this check box selects the current update set from the Update Set picker if the picker is enabled and the update set is in the In Progress state.
Description	Enter a description of the update set.

Select an Update Set

Update sets track changes as you develop. Follow this procedure to specify the update set that should collect the updates that you make:

1. Open the **Settings** panel.
2. Select the desired update set from the Update Set picker. Starting with the Fuji release, only update sets that are appropriate for the current application scope appear in the selection list and the associated application is displayed in square brackets.



Settings

What's new

Home

System Administration

Home

Preferences

Wrap Longer Text in List Columns

Compact View

Accessibility Enabled

Loading

With the Form | After Form Loads | On-Demand

Application

Incident App

Update Set

Default [Incident App]

✓ Incident Release 1 [Incident App]

Time zone

US/Pacific-New

Update Set picker

Now, any customization that you perform on a tracked table is recorded by the update set. For example:

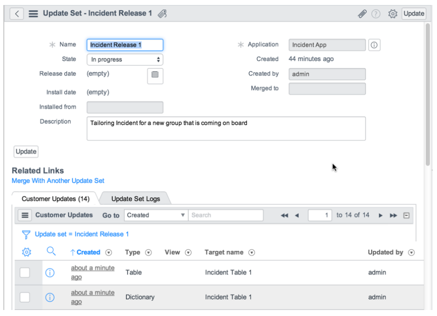
- An update record is added or updated in the current update set.
- A new version record is created with the current update set as the source.
- You can compare versions of any customized object, and you can revert changes to an older version.

Viewing Changes in an Update Set

Follow this procedure to see the customizations that make up an update set:

1. Navigate to **System Update Sets > Local Update Sets**.
2. Select the update set name.
3. View the **Customer Updates** related list.

   You can compare any update to the current version. Right-click the update record and select **Compare to Current**.



Update Set - Incident Release 1

Name	Incident Release 1	Application	Incident App
State	In progress	Created	44 minutes ago
Release date	(empty)	Created by	admin
Install date	(empty)	Merged to	
Installed from			
Description	Tailoring Incident for a new group that is coming on board		

Related Links

Merge With Another Update Set

Customer Updates (14) | Update Set Logs

Customer Updates Go to Created 1 to 14 of 14

Update set = Incident Release 1

Created	Type	View	Target name	Updated by
about a minute ago	Table		Incident Table 1	admin
about a minute ago	Dictionary		Incident Table 1	admin

Update set with addition of the 'Incident Table' table

Viewing Customer Update Records

The Customer Update [sys_update_xml] table contains one record per customized object. From the Customer Update record you can determine:

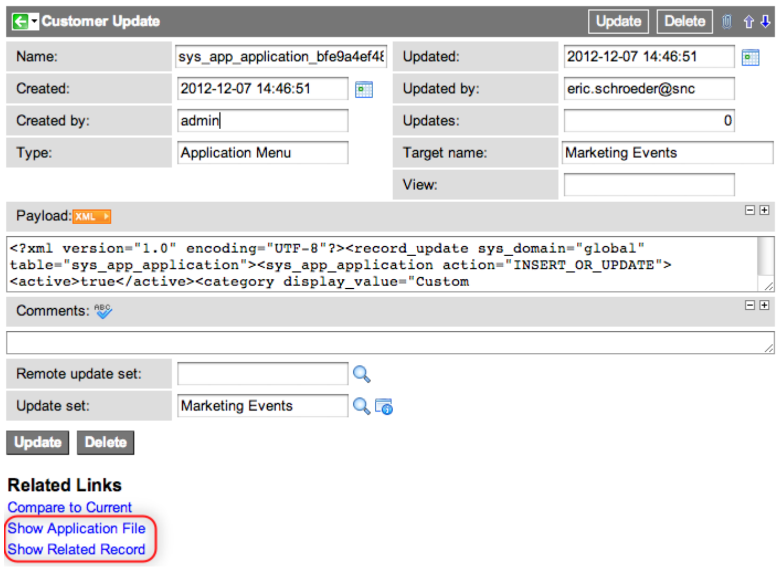
- The update set containing the customized object
- The type of action applied to the customized object
  - INSERT
  - INSERT_OR_UPDATE
  - UPDATE
  - DELETE
- The type of object customized
- The target object of the update
- The Sys ID of the customized object (if it is a change to a particular record)
- The user who customized the object
- The date and time the object was customized

Navigating Updated Records

You can navigate between a customer update record and the customized object or the application file for the object (starting with the Calgary release).

Navigate from an update record to:

- The customized object, such as the application menu record, by clicking the **Show Related Record** related link.
- The application file record for the object by clicking the **Show Application File** related link.



Customer Update

Name:	sys_app_application_bfe9a4ef4	Updated:	2012-12-07 14:46:51
Created:	2012-12-07 14:46:51	Updated by:	eric.schroeder@snc
Created by:	admin	Updates:	0
Type:	Application Menu	Target name:	Marketing Events
		View:	

Payload:

```
<?xml version="1.0" encoding="UTF-8"?><record_update sys_domain="global"
table="sys_app_application"><sys_app_application action="INSERT_OR_UPDATE">
<active>true</active><category display_value="Custom
```

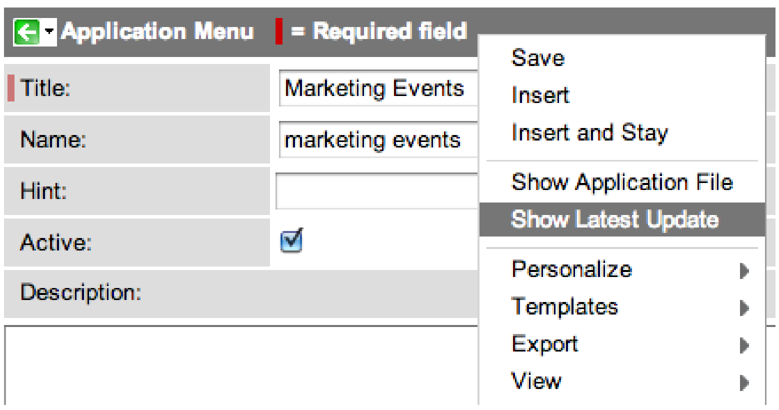
Comments:

Remote update set:	
Update set:	Marketing Events

Related Links
Compare to Current
Show Application File
Show Related Record

Show Related Record

Navigate from a customized object or an application file to the current customer update record by right-clicking the form header and selecting **Show Latest Update**.



Application Menu = Required field

Title:	Marketing Events
Name:	marketing events
Hint:	
Active:	✓
Description:	

Show Latest Update

Reporting on Updates

Run a report to view the customizations and configuration changes for an instance.

To view update set reports:

1. Navigate to **Reports > View / Run** and locate the **Customer Update** section.
2. Run any of the available reports or create a new report. The following reports are available:

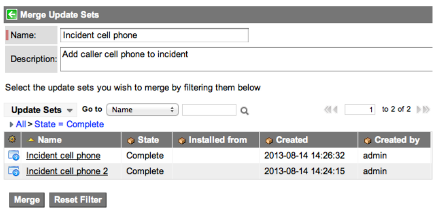
- **Application Changes (Incident)**: Displays all changes made to the Incident table. Select a different table and run the report again to view all changes to another application.
- **My Changes**: Displays all changes created or updated by the current user, grouped by table name.

Merging Update Sets

You can merge multiple update sets into one update set to simplify the transfer process. To merge update sets:

1. Navigate to **System Update Sets > Merge Update Sets**. By default, the list is filtered to show only update sets that are in progress.

   Alternatively, navigate to **System Update Sets > Merge Completed Sets**. By default, the list is filtered to show only update sets that are in the **Complete** state. For example, you may want to use this filter after pushing changes or transferring update sets from a development to a test instance.
2. Filter the list to show only the update sets that you want to merge.



Merge Update Sets

Name:	Incident cell phone
Description:	Add caller cell phone to incident

Select the update sets you wish to merge by filtering them below

Update Sets Go to Name 1 to 2 of 2

All > State = Complete

Name	State	Installed from	Created	Created by
Incident cell phone	Complete		2013-08-14 14:26:32	admin
Incident cell phone 2	Complete		2013-08-14 14:24:15	admin

Merge update sets

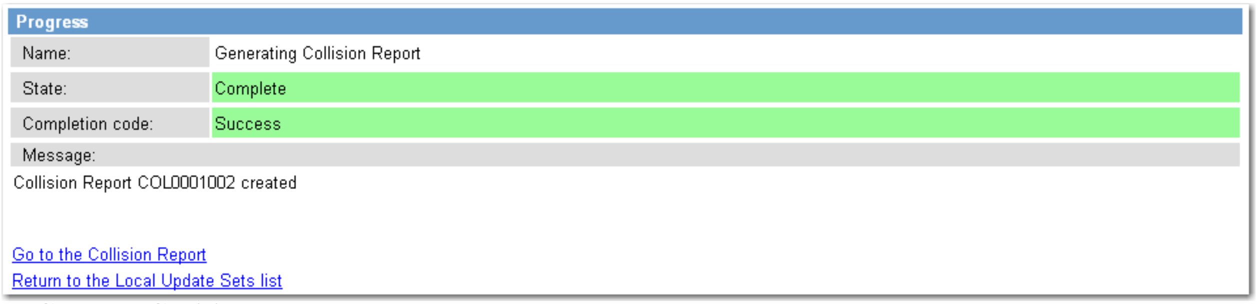
3. Enter a **Name** for the new update set. Updates are added to this set when the original sets are merged.
4. [Optional] Enter a **Description** for the update set.
5. Click **Merge**.
6. In the confirmation dialog box, click **OK**.
   - The new update set is created.
   - The most recent change for each object is moved from the original sets to the new set. Only changes that are not merged into the new set remain in the original sets. A message indicates how many updates were moved and how many were skipped. For example, if both update sets modify the Incident form, only the most recent change is moved to the new update set. The other modification remains in its original update set to provide a record of the changes that were not moved.
7. [Recommended] Verify that the correct changes were moved to the new set by scrolling down to the **Merged Update Sets** related list and opening the old update set records.
8. [Recommended] Delete or empty the original update sets to avoid committing an older change by mistake. The system does not remove updates that were not merged into the new set.

Comparing Local Update Sets

Administrators can preview local and remote (retrieved) update sets and compare these sets with one another to resolve conflicting changes. Compare local update sets to identify collisions and ensure that the proper changes are being committed. Resolve all conflicts before moving an update set between instances.

1. Navigate to **System Update Sets > Local Update Sets**.
2. Select the check boxes beside the update sets to compare.
3. In the **Action** choice list, select **Compare Update Sets**.

   The progress screen appears as ServiceNow generates the collision report.



Progress

Name:	Generating Collision Report
State:	Complete
Completion code:	Success
Message:	

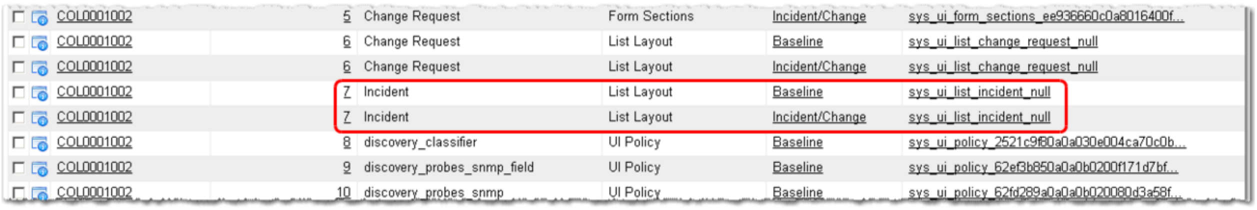
Collision Report COL0001002 created

Go to the Collision Report

Return to the Local Update Sets list

4. Click **Go to the Collision Report** when the report is complete.

   The Update Set Collisions list appears, showing all the changes in the selected sets.
5. Inspect the list for collisions by locating duplicate **Collision Numbers** that show the same change in separate update sets.



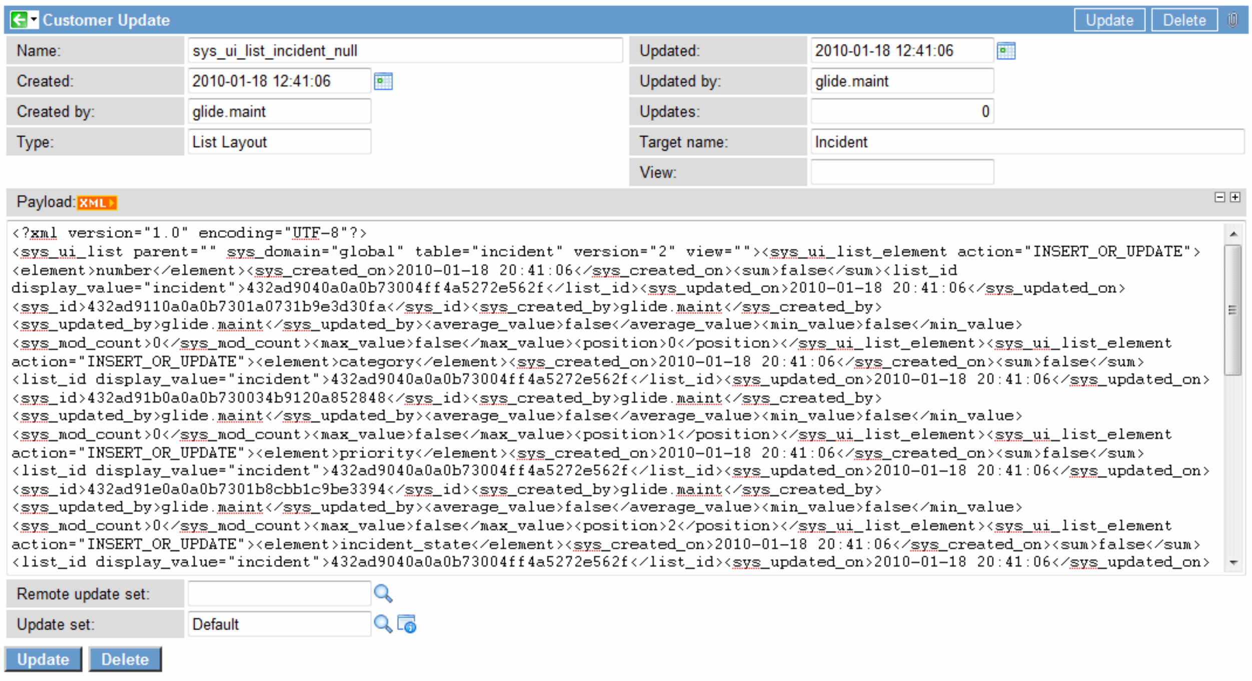
☐	COL0001002	5	Change Request	Form Sections	Incident/Change	sys_ui_form_sections_ee936660c0a8016400f...
☐	COL0001002	6	Change Request	List Layout	Baseline	sys_ui_list_change_request_null
☐	COL0001002	6	Change Request	List Layout	Incident/Change	sys_ui_list_change_request_null
☐	COL0001002	7	Incident	List Layout	Baseline	sys_ui_list_incident_null
☐	COL0001002	7	Incident	List Layout	Incident/Change	sys_ui_list_incident_null
☐	COL0001002	8	discovery_classifier	UI Policy	Baseline	sys_ui_policy_2521c9f80a0a030e004ca70c0b...
☐	COL0001002	9	discovery_probes_snmp_field	UI Policy	Baseline	sys_ui_policy_62ef3b850a0a0b0200f171d7bf...
☐	COL0001002	10	discovery_probes_snmp	UI Policy	Baseline	sys_ui_policy_62fd289a0a0a0b020080d3a58f...

6. Resolve the collision by deleting the unwanted update record from one of the update sets.

   a. Click the link in the **Sys update** column for the unwanted update (`sys_ui_list_incident_null` in the example).

   b. Click **Delete**.

   Note: You must open the update record to delete the record. You cannot delete the update by selecting the check box beside the entry in the Update Set Collisions list and using the **Delete** action. When you delete the update record, the customization is *not* backed out of the instance. Only the record of the customization is deleted.



Customer Update

Name:	sys_ui_list_incident_null	Updated:	2010-01-18 12:41:06
Created:	2010-01-18 12:41:06	Updated by:	glide.maint
Created by:	glide.maint	Updates:	0
Type:	List Layout	Target name:	Incident
		View:	

Payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<sys_ui_list parent="" sys_domain="global" table="incident" version="2" view=""><sys_ui_list_element action="INSERT_OR_UPDATE">
<element>number</element><sys_created_on>2010-01-18 20:41:06</sys_created_on><sum>false</sum><list_id
display_value="incident">432ad9040a0a0b73004ff4a5272e562f</list_id><sys_updated_on>2010-01-18 20:41:06</sys_updated_on>
<sys_id>432ad9110a0a0b7301a0731b9e3d30fa</sys_id><sys_created_by>glide.maint</sys_created_by>
<sys_updated_by>glide.maint</sys_updated_by><average_value>false</average_value><min_value>false</min_value>
<sys_mod_count>0</sys_mod_count><max_value>false</max_value><position>0</position></sys_ui_list_element><sys_ui_list_element
action="INSERT_OR_UPDATE"><element>category</element><sys_created_on>2010-01-18 20:41:06</sys_created_on><sum>false</sum>
<list_id display_value="incident">432ad9040a0a0b73004ff4a5272e562f</list_id><sys_updated_on>2010-01-18 20:41:06</sys_updated_on>
<sys_id>432ad91b0a0a0b730034b9120a852848</sys_id><sys_created_by>glide.maint</sys_created_by>
<sys_updated_by>glide.maint</sys_updated_by><average_value>false</average_value><min_value>false</min_value>
<sys_mod_count>0</sys_mod_count><max_value>false</max_value><position>1</position></sys_ui_list_element><sys_ui_list_element
action="INSERT_OR_UPDATE"><element>priority</element><sys_created_on>2010-01-18 20:41:06</sys_created_on><sum>false</sum>
<list_id display_value="incident">432ad9040a0a0b73004ff4a5272e562f</list_id><sys_updated_on>2010-01-18 20:41:06</sys_updated_on>
<sys_id>432ad91e0a0a0b7301b8cbb1c9be3394</sys_id><sys_created_by>glide.maint</sys_created_by>
<sys_updated_by>glide.maint</sys_updated_by><average_value>false</average_value><min_value>false</min_value>
<sys_mod_count>0</sys_mod_count><max_value>false</max_value><position>2</position></sys_ui_list_element><sys_ui_list_element
action="INSERT_OR_UPDATE"><element>incident_state</element><sys_created_on>2010-01-18 20:41:06</sys_created_on><sum>false</sum>
<list_id display_value="incident">432ad9040a0a0b73004ff4a5272e562f</list_id><sys_updated_on>2010-01-18 20:41:06</sys_updated_on>
```

Remote update set:	
Update set:	Default

7. Run the comparison again to make sure all the collisions have been resolved.

Understanding Collisions

A collision is an update that has a newer local update. ServiceNow detects collisions by comparing the values in the **Name** and **Updated** fields of the Customer Update record from each update set. If the name matches but there are different update date values, then there is a collision.

Coalesce Strategies

Update sets can detect collisions between identical records that you independently create on separate instances. To detect such collisions, the record must have a coalesce strategy based on coalescing columns. Because collision detection depends on uniqueness of tables, the tables must be unique when the coalescing columns are combined. Records that are not listed here will not collide if the same record is created separately on different instances.

Type	Coalescing Columns
sys_db_object	name
sys_dictionary	name, element
sys_choice_set	name, element, language
sys_documentation	name, element, language
sys_properties	name
sys_report_chart_color	name, element, value
sys_ui_form Starting with Fuji	name, view, sys_domain
sys_ui_message	documentkey, language
sys_ui_list Starting with Fuji	name, view, sys_domain, element, relationship, parent
sys_ui_section Starting with Fuji	name, view, caption, sys_domain
sys_ui_related_list Starting with Fuji	name, view, related_list, sys_domain
sys_ui_view	name
sys_user_role	name
sys_wizard Starting with Fuji	name

Matching Customer Update Record Names for Collisions

To understand coalescing, it helps to understand how records that do not coalesce work:

For most record types, when a Customer Update is moved to a new instance, the system does not detect collisions for the following reason:

> When you create a record, it receives a unique Sys ID. For most record types, the Sys ID becomes part of the Customer Update record name. For example:
>
> `sysevent_email_template_9e1998c078b71100a92ecacd80df1d39`
>
> Creating an identical record in the same table on another instance produces a Customer Update record name with a different Sys ID, for example:
>
> `sysevent_email_template_10b958c8653311005840134572f8e020`
>
> As a result, even though the records might be otherwise identical, the records have different names so the system does not detect the collision.

Coalescing records, in contrast, use the following approach to naming records and determining collisions:

The following Customer Update record types use some or all of their coalescing columns instead of the Sys ID in their names. The resulting identical record name in each instance helps the system to identify collisions even if the records have different Sys IDs.

> sys_dictionary
>
> sys_documentation
>
> sys_choice_set
>
> sys_ui_list
>
> sys_ui_related_list

When a customer update is moved from one instance to another, it may be re-written to match the target instance. The re-write can involve changing the update name of the customer update and one or more sys_ids within the update. The re-writes are done when the record or the reference field is for a table that uses a coalesce strategy. This ensures that the customer update will be applied to the correct record. For example, if the sys_dictionary record for tablename.fieldname has sys_id "123456789" on instance A and sys_id "987654321" on instance B, when a customer update that refers to that record is retrieved from instance A and recorded in the sys_update_xml table on instance B, references to "123456789" are updated to read "987654321".

Preventing Duplicate Records

To prevent creating duplicate records with update sets, you can:

- Move data with update sets rather than recreating it on separate instances to ensure the records have the same Sys ID..
- Export and import records as XML files to ensure the records have the same Sys ID.
- Enable a unique index for the table from the system dictionary.

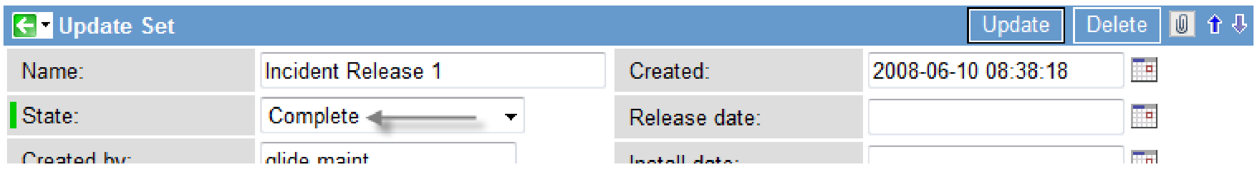
Note: *The default records included in the baseline system will always have the same Sys ID because the instance imports the records as XML files during instance provisioning.*

Mark an Update Set as Complete

When you have completed the customizations and compared local update sets to resolve conflicts, mark the update set as **Complete**.

Note: *Mark an update set as Complete only when it is ready to migrate. Once an update set is complete, do not change it back to In progress. Instead, create another update set for the rest of the changes, and be sure to commit them together in the order that they were created. Naming conventions may help in this case (for example, Performance Enhancements and Performance Enhancements 2).*

1. Open the update set record.
2. Change the **State** of the update set from **In progress** to **Complete**.
   - The update set is available for other instances to retrieve. See Transferring Update Sets.
   - No additional customizations are tracked in the update set.



Name:	Incident Release 1	Created:	2008-06-10 08:38:18
State:	Complete	Release date:	
Created by:	glide.maint	Install date:	

References

[1] https://docs.servicenow.com/bundle/jakarta-application-development/page/build/system-update-sets/concept/update-set-procedures.html

Transferring Update Sets

Note: *This article applies to Fuji and earlier releases. For more current information, see Update Set Transfers* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.'**

Overview

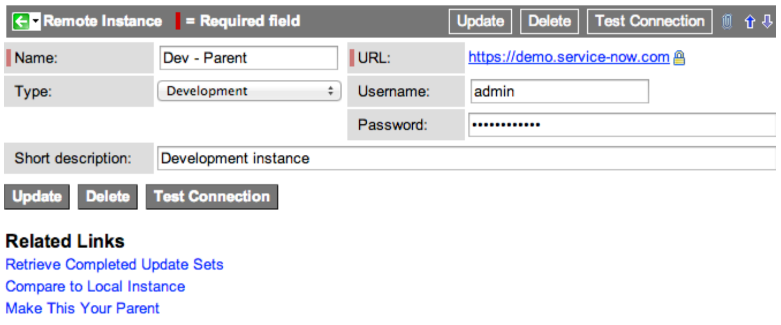
When an update set is completed, you can transfer it to another instance. Transfer update sets between instances to move customizations from development, through testing, and into production.

Note: *Properties that are tagged as Private are excluded from update sets. Keeping system properties private prevents settings in one instance from overwriting values in another instance. For example, you may not want a system property in a production instance to use a particular value from a development instance. See Adding a Property.*

Retrieving Update Sets

To retrieve completed update sets from another instance:

1. If IP address access control is enabled on the source instance, set up the target instance as an exception.
2. On the target instance, navigate to **System Update Sets > Update Source** and click **New**.
3. Define the connection settings (see table).



Name:	Dev - Parent	URL:	https://demo.service-now.com
Type:	Development	Username:	admin
		Password:	••••••••••••
Short description:	Development instance		

Define connection settings

4. Click **Test Connection** (starting with the Dublin release).
   - If the connection is successful, a confirmation message appears.
   - If the connection fails, a warning message identifies the reason the connection failed.
5. If the connection fails, modify the settings to establish connectivity.
   - You must establish connectivity before you can save the connection settings (starting with the Dublin release).
   - You may want to modify the source instance (for example, change the password).
6. Right-click the form header and select **Save**.
7. Under **Related Links**, click **Retrieve Completed Update Sets**.
   - Any update sets marked as **Completed** are transferred from the source instance to the target instance. Update sets that already exist on the target instance are skipped.
   - The confirmation page provides detailed messages about how many update sets were transferred and how many were skipped (starting with the Dublin release).
   - To view retrieved update sets, navigate to **System Update Sets > Retrieved Update Sets**.

Field	Description
Name	Enter a unique name describing the instance.
Type	Specify whether the remote instance is a development, test, or UAT instance.
Active	Specify whether the local instance can transfer update sets to the remote instance. You can only transfer update sets to active remote instances (starting with the Eureka release).
URL	Specify the URL of the remote instance using the appropriate transfer protocol. Each remote instance record should have a unique URL. Creating duplicate records with the same URL can cause errors. The remote instance must be on the same release family as the local instance (starting with the Eureka release). **Note:** You cannot change the URL after the system verifies the connection (starting with the Dublin release). Use the **Active** field to deactivate unwanted remote instances.
Username	Enter the user on the remote instance who authorizes transferring update sets to this the instance. This user account must have the admin user role on the remote instance.
Password	Enter the password of the authorizing user.
Short description	[Optional] Enter any other relevant information about the remote instance.

Transferring with IP Access Control

If IP address access control is enabled on the source instance or the source instance resides in a different datacenter than the target instance, complete the following steps before transferring an update set:

1. Go to this URL [2] to find out the IP addresses of all application nodes supporting your instance. You can also contact Customer Support.
2. On the source instance, navigate to **System Security > IP Address Access Control**.
3. Add the IP address from step one as an exception.

Transferring with Basic Authentication

If the source instance has basic authentication turned on for SOAP requests, you must use valid credentials to retrieve update sets.

Transferring with an XML File

You can also unload an update set as an XML file and then transfer it to another instance. See Saving Customizations in a Single XML File.

Previewing Remote Update Sets

Previewing compares an update set retrieved from a remote instance to updates on the local instance and detects potential problems. You must preview an update set and address all problems before you can commit it.

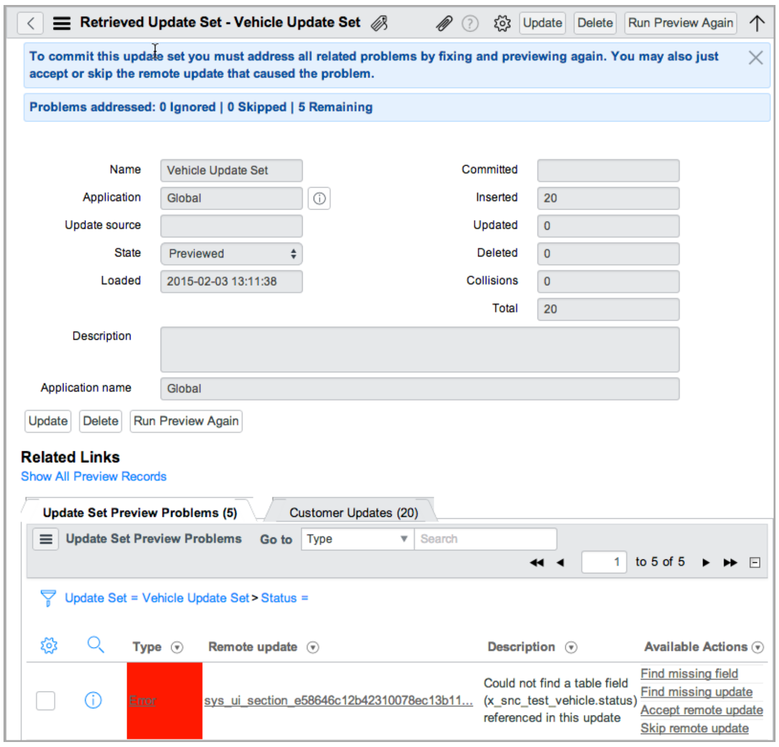
To preview a retrieved update set:

1. Navigate to **System Update Sets > Retrieved Update Sets**.
2. Open the update set.
3. Click **Preview Update Set**.

   The **Update Set Preview** page shows results and lists problems. Read the information and click **Close**.
4. On the **Retrieved Update Set** form:

If no problems were detected, click **Commit Update Set** to commit all changes on the instance without reviewing the preview results. See Committing Update Sets.

If problems were detected, resolve each problem in the Update Set Preview Problems related list.



To commit this update set you must address all related problems by fixing and previewing again. You may also just accept or skip the remote update that caused the problem.

Problems addressed: 0 Ignored | 0 Skipped | 5 Remaining

Name	Vehicle Update Set	Committed	
Application	Global	Inserted	20
Update source		Updated	0
State	Previewed	Deleted	0
Loaded	2015-02-03 13:11:38	Collisions	0
		Total	20

Description

Application name: Global

Update Set = Vehicle Update Set > Status =

Type	Remote update	Description	Available Actions
Error	sys_ui_section_e58646c12b42310078ec13b11...	Could not find a table field (x_snc_test_vehicle.status) referenced in this update	Find missing field Find missing update Accept remote update Skip remote update

Update set preview detects problems

5. [Optional] Open the update set record and click **Show All Preview Records** to make sure the correct updates are being committed. See Reviewing Preview Records.
6. [Optional] Open the update set record and click **Run Preview Again** to run the comparisons again. Review the preview results again to ensure that all problems are resolved.

Resolving Preview Problems

The process of previewing an update set detects problems that may occur if you commit the updates on the local instance. Before you commit an update set, follow this procedure to resolve all of the problems that the preview process discovered.

1. Navigate to **System Update Sets > Retrieved Update Sets**.
2. Open the update set record and scroll down to the **Update Set Preview Problems** related list.
3. Review each problem description to determine the cause and resolve the problem as described in the table. Alternatively, choose one of the following options:
   - **Accept remote update:** commit the remote update set without fixing the problem.
   - **Skip remote update:** skip the update when you commit the update set.

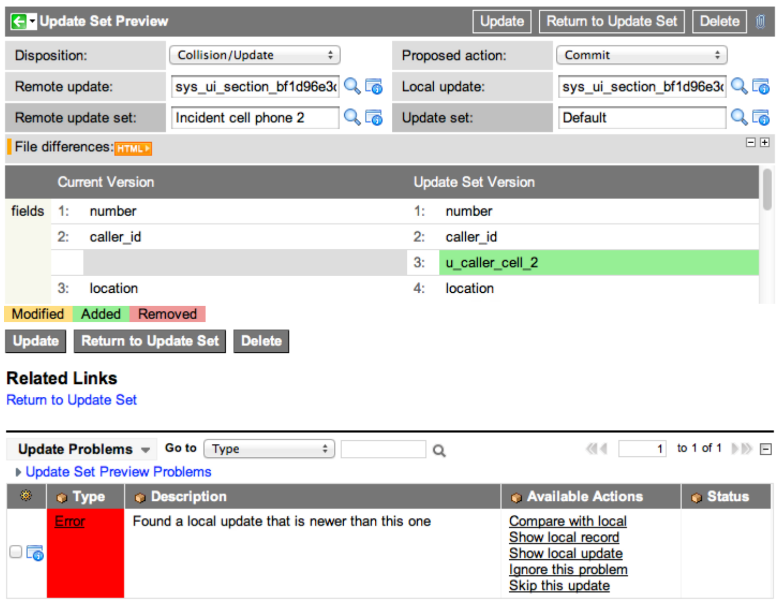
Type and Example	Description	Resolution
Missing object Example: Could not find a record in sys_ui_policy referenced in this update	The object or a referenced object does not exist on the target instance. For example: • An update modifies the form layout for a table that has not been created in the local instance. • A UI policy action is included in the update set, but the parent UI policy is not.	Create another update set on the source instance to transfer the missing object to the local instance or create the object on the local instance. Use the following **Available Actions** to assist in problem resolution: • **Find missing field** or **Find missing record:** Opens a new window and searches the source instance for the missing field (dictionary entry) or record. • **Find missing update:** Opens a new window and searches the source instance for the update record that corresponds to the missing field or record.

Type mismatch (Calgary release and earlier) Example: Found a table field (u_case.due_date), but with a different type. (Found date, expected string.)	The type of a column on the target instance does not match the type in the update set. For example, the target instance has a field called **due_date** with a type of **date** and the retrieved set has the same field with a type of **string**.	Change the field type on the source instance and transfer the update set again. Alternatively, if the type change is intentional, manually change the data type on the target instance. Use these **Available Actions** to assist in problem resolution: • **Show local field** or **Show local record:** opens the dictionary entry or record on the local instance. • **Show local update:** opens the update record on the local instance that corresponds to the dictionary entry or record. **Note:** *A data type change that results in data loss is skipped automatically, even if you accept the update. See the update log.*
Collision Example: Found a local update that is newer than this one	A change in the update set is older than a change to the same object on the local instance.	Compare the two updates and determine which version to use. To use the version on the local instance, select **Skip remote update**. To use the version in the update set, select **Accept remote update**. Use these **Available Actions** to assist in problem resolution: • **Compare with local:** opens the preview record, which provides a comparison of the differences between the local version and the version in the update set. • **Show local field** or **Show local record** • **Show local update**
Uncommitted update Example: Could not find a table field (u_case.u_reference) referenced in this update, but did find it in another uncommitted update set	The object exists in another remote update set that has not been committed.	Commit the other remote update set first or move this update to the other update set. Use these **Available Actions** to assist in problem resolution: • **Show uncommitted update:** opens the update record in the other remote update set. • **Show uncommitted update set:** open the other remote update set record.
Application scope validation issue	The previewer identifies the following combination of states as a problem: The scope for the update set is not Global and The application is not found on the target instance and The application is not included with the update set and The application is not found on the ServiceNow Store.	Transfer the update set only to instances with that include the application scope or ensure that the update set includes the application.

Reviewing Preview Records

Previewing an update set creates a preview record for each update. You can review the preview records to make sure the correct updates are being committed.

1. Preview the update set.
2. Open the update set record, and click **Show All Preview Records**.
3. Open a preview record and review the information (see table).


Update Set Preview

Disposition:	Collision/Update	Proposed action:	Commit
Remote update:	sys_ui_section_bf1d96e3c	Local update:	sys_ui_section_bf1d96e3c
Remote update set:	Incident cell phone 2	Update set:	Default

File differences:

	Current Version		Update Set Version	
fields 1:	number	1:	number	
2:	caller_id	2:	caller_id	
		3:	u_caller_cell_2	
3:	location	4:	location	

Modified Added Removed

Related Links

Return to Update Set

Update Problems

Update Set Preview Problems

Type	Description	Available Actions	Status
Error	Found a local update that is newer than this one	Compare with local Show local record Show local update Ignore this problem Skip this update	

Preview record

4. If necessary, address any problems listed in the **Update Problems** related list.
5. In the **Proposed action** field, select the action to take when committing the update set.
6. Click **Update** to save the action.
7. Repeat steps 3 – 7 for every preview record you want to review.

Field	Description
Disposition	Indicates when a collision is detected: • **Collision/Update**, **Collision/Insert**, or **Collision/Delete:** the change is *older* than a change to the same object on the local instance. • **Update**, **Insert**, or **Delete:** the change does not conflict with a change on the local instance.
File differences	Compares the most recent version of the object on the local instance with the version in the update set. Differences are marked with a color key. Deletions are highlighted in red, additions in green, and modifications in yellow.
Proposed action	Indicates how to handle the change when the update set is committed. • **Commit:** accept the change in the remote update. The default proposed action for every preview record is **Commit**, even if a newer update exists on the instance. • **Skip:** reject the change.

Committing Update Sets

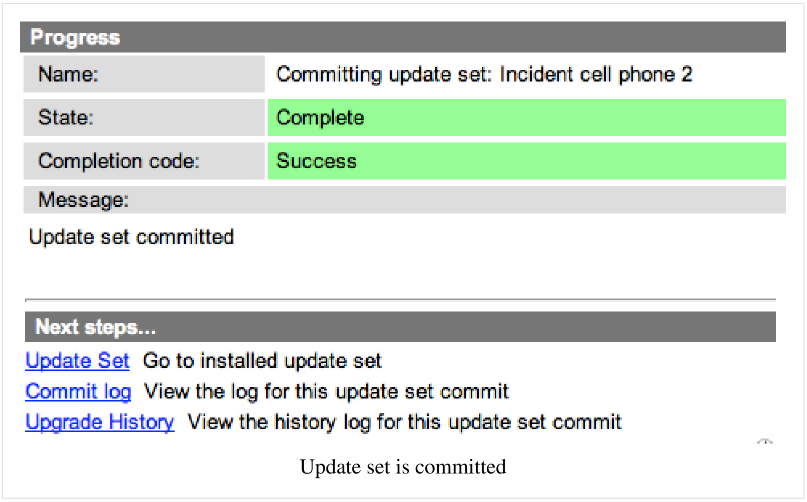
Warning: Avoid committing an update set during business hours because it can affect instance performance.

When you have previewed the update set and resolved any conflicts or problems, commit the update set. Committing an update set applies all changes to the instance and creates a local copy of the update set that contains an update record for every change.

1. Navigate to **System Update Sets > Retrieved Update Sets** and open the update set.
2. Address any problems. You cannot commit an update set until all problems are addressed.
3. Click **Commit Update Set**.
4. If any updates have a proposed action of **Skip**, a confirmation message appears when you commit the update set. (Calgary release only.)
   - Click **Cancel** to return to the preview and reevaluate the change. None of the updates are committed.

- Click **OK** to skip the change and continue committing the changes that are marked as **Commit**.

  A completion page appears when the update set has been successfully committed.



Progress

Name:	Committing update set: Incident cell phone 2
State:	Complete
Completion code:	Success
Message:	

Update set committed

Next steps...

Update Set Go to installed update set
Commit log View the log for this update set commit
Upgrade History View the history log for this update set commit

Update set is committed

5. [Recommended] Click **Commit log** on the confirmation page, or navigate to **System Update Sets > Update log** and filter for the update set name.
   - Look for warnings that contain the text **unsafe edit**. The system automatically skips any changes that will result in data loss, such as changing the type of a field that contains data. You must manually make any of these changes, if necessary. Use caution when making changes that affect production data.
   - Look for errors that indicate which records failed to commit and why. Create a new update set to address those failures, if necessary.
6. [Recommended] When you are no longer working on the update set but do not want it to be transferred to another instance, navigate to **System Update Sets > Local Update Sets** and open the local update set record. Change the **State** to **Ignore**.

   For completed update sets on the production instance, you should always change the state to **Ignore**. This state ensures the update set is not committed again when cloning the instance.

Backing Out Update Sets

You can back out changes to existing records for any committed update set, starting with the Fuji release. Previously, you could back out only the most recently committed local update set.

⚠	**Warning:** Do not back out the **Default** update set because that can damage the configuration of the instance.

1. Navigate to **System Update Sets > Local Update Sets** or **System Update Sets > Retrieved Update Sets**.

   On the Eureka release, navigate to **System Update Sets > Local Update Sets**.
2. Open the update set record.
3. Carefully review the contents of the update set and consider whether there will be problems if it is backed out. See the table of expected results.

   If backing out an update set is not sufficient or will cause issues, then, instead, create and commit a new update set to reverse the customizations.
4. Click **Back Out**. The **Progress** popup displays actions, progress, and problems. Problems are changes in more recent update sets that affect the update set that is being backed out. You receive a warning for each problem that is found and you must take action on each problem before proceeding with back out.
   - If you choose to keep the latest version, click **Use Current**.
   - If you choose to back out to the previous version, click **Back Out**.

All changes are reversed as described in the table. The current update set tracks all of the new changes that occur.

The update set and all associated update records are deleted. If needed, you can still navigate to the retrieved update set, preview it, and commit it again.

Note: *If you commit, back out, and then reapply a remote update set, errors appear in the previewer because backing out an update set creates 'delete' updates in the current update set. The deletes are considered more recent changes and cause collisions.*

Expected Results of the Back Out Process

The back out process reverses both record-level updates and changes to the dictionary, as described in the following table:

Warning: Some changes cause loss of data.

Customer Update	Back out action
A new table	The table is dropped from the database, deleting any data from it.
A new field	The field is dropped from the database, deleting any data from it.
A deleted field	The field is restored to the database, but the original data is lost.
A resized field	The field resize is reversed. If the field has been increased, data is truncated first to avoid errors.
A configured form	The form is reverted to its previous state.
A record is inserted	The record is deleted.
A record is deleted	The record is restored with its original data.

Deleting Update Sets

To undo customizations, back out the update set.

Administrators can delete an update set only when it is not the current update set and it is empty (no `sys_update_xml` entries are associated with it). For example, after merging update sets you may choose to delete the original sets. This function is restricted by an access control rule (ACL) on the Update Set [sys_update_set] table.

Deleting update sets that contain sys_update_xml entries is restricted and not recommended because:

- It does not undo the updates.
- It removes any record of who applied the customizations.
- It removes the `sys_update_xml` entries associated with the update set, so customizations are overwritten when the instance is upgraded.

Deleting Update Entries

Deleting `sys_update_xml` entries is not recommended because:

- It removes the record of modifications to the instance.
- Your customizations may be overwritten when the instance is upgraded.

When you try to delete an update entry, a warning message appears. Click **OK** to confirm the deletion.

References

[1] https://docs.servicenow.com/bundle/jakarta-application-development/page/build/system-update-sets/reference/update-set-transfers.html
[2] https://hi.service-now.com/com.glideapp.servicecatalog_cat_item_view.do?sysparm_id=b85d23ae6f086100a5117357ea3ee4f3

Saving Customizations in a Single XML File

Note: *This article applies to Fuji and earlier releases. For more current information, see Save an Update Set as a Local XML File* [1] *at* http://docs.servicenow.com **The ServiceNow Wiki is no longer being updated. Visit** http://docs.servicenow.com **for the latest product documentation.'**

Overview

An update set is an XML file containing a list of changes to an instance. Administrators can save an update set XML as a local file that can be shared with another instance. Typically you create an XML file when one of these conditions apply.

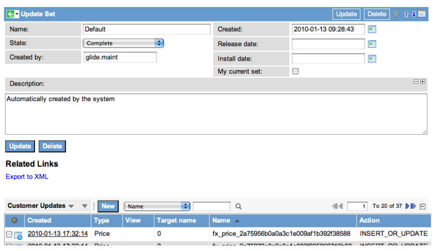
- The two instances do not have *network connectivity* so you cannot retrieve update sets from the remote instance nor create a data source to pull data directly from the source instance
- You do not want to provide *administrator credentials* to the source instance (for example, you do not want to share an administrator password with people outside your company) so you cannot retrieve update sets nor create a data source
- You want to *back up* important customizations locally

Saving Customizations to a Single XML

To save an update set an an XML file:

1. Navigate to a completed Local Update Set or Retrieved Update Set.
2. Click the link **Export to XML**.
3. Save the XML file.

An XML file is created. When this file is uploaded to another instance, it appears as a Retrieved Update Set regardless of whether it is local or retrieved on the instance where it is created.



Name:	Default	Created:	2010-01-13 09:28:43
State:	Complete	Release date:	
Created by:	glide.maint	Install date:	
		My current set:	

Description:

Automatically created by the system

Related Links

Export to XML

Created	Type	View	Target name	Name	Action
2010-01-13 17:32:14	Price		0	fx_price_2a75956b0a0a3c1e009af1b392f38588	INSERT_OR_UPDATE

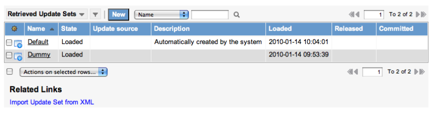
Export to XML

Loading Customizations from a Single XML File

To load an update set from an XML file:

1. Elevate privileges to the security_admin role.
2. Navigate to **System Update Sets > Retrieved Update Sets**.
3. Click the link **Import Update Set from XML**.
4. Click **Choose File** and select an XML file.
5. Click **Upload**.

The customization is now available as a Retrieved Update Set with state **Loaded**. Follow standard procedure to commit the update set.



Name	State	Update source	Description	Loaded	Released	Committed
Default	Loaded		Automatically created by the system	2010-01-14 10:04:01		
Dummy	Loaded			2010-01-14 09:53:39		

Related Links

Import Update Set from XML

Import from XML

How it Works

The ability to export and import customizations as an XML file is provided by the following UI Actions:

- **Export to XML** on the table **sys_update_set**
- **Export to XML** on the table **sys_remote_update_set**
- **Import Update Set from XML** on the table **sys_remote_update_set**

The Export to XML UI Action on **sys_update_set** calls a processor called **UnloadRetrievedUpdateSet**, which transforms a local update set into a retrieved update set, exports the retrieved update set with its related list, and then deletes the temporary update set if necessary.

Both Export to XML UI actions depend on the Script Include **ExportWithRelatedLists**, which exports a record and manually defined related lists to a single XML.

References

[1] https://docs.servicenow.com/bundle/jakarta-application-development/page/build/system-update-sets/task/t_SaveAnUpdateSetAsAnXMLFile.html

Related Topics

Moving Workflows with Update Sets

Note: *This article applies to Fuji. For more current information, see Workflow Movement with Update Sets* [1] *at* http://docs.servicenow.com The ServiceNow Wiki is no longer being updated. Please refer to http://docs.servicenow.com for the latest product documentation.

Overview

ServiceNow tracks workflows in update sets differently than other records because workflow information is stored across multiple tables. Changes made to a workflow version are not added to the update set until the workflow is published, at which point the entire workflow is added into the update set. For additional information about update sets, see:

- System Update Sets
- ValidateUpdateSetDependencies

Tables and Versions

Update sets store workflows as a single Workflow [wf_workflow] record and only retain the latest version with the update type of **Workflow**, starting with the Fuji release.

Workflow Update Set Migration

Examine the following use cases to avoid common mistakes when moving a workflow with an update set:

- Simple
- Subflow Dependency (Success)
- Subflow Dependency (Failure)
- Subflow Dependency (Risk)

Use Case - Simple

Create a new workflow with no dependencies, and then migrate the workflow in an update set.

1. User A selects Update Set A.
2. User A creates a new workflow called Workflow A.
3. User A publishes Workflow A.

   A customer update set record is added to Update Set A containing an XML payload, including the published Workflow A and all activity dependencies. The XML payload also contains the workflow input variables associated with the workflow.
4. User A completes Update Set A and migrates it to the production instance.
5. Update Set A commits successfully.
6. Workflow A works as expected.

Use Case - Subflow Dependency (Success)

Successfully edit and migrate an existing workflow and its dependent subflow.

1. User A selects Update Set B.
2. User A checks out Workflow A.
3. User A adds a subflow called Workflow B to Workflow A.

   Assume that Workflow B was previously published and migrated to the production instance.
4. User A publishes Workflow A.

   A customer update set record is added to Update Set B containing an XML payload, including the published Workflow A and all activity dependencies. The XML payload also contains the workflow input variables associated with the workflow.
5. User A completes Update Set B and migrates it to the production instance.
6. Update Set B commits successfully.
7. Workflow A works as expected with Workflow B as a subflow.

Use Case - Subflow Dependency (Failure)

Edit and migrate an existing workflow from a test instance to a production instance that fails to run on the production instance because of a missing dependent subflow.

1. User A selects Update Set C.
2. User A checks out Workflow A.
3. User A adds a subflow called Workflow B to Workflow A.

   Assume that Workflow B was previously published, but has *not* been migrated to the production instance.
4. User A publishes Workflow A.

   A customer update set record is added to Update Set C containing an XML payload, including the published Workflow A and all activity dependencies. The XML payload also contains the workflow input variables associated with the workflow.

   Notably absent from Update Set C, is the subflow called Workflow B. Workflow B was published before User A selected Update Set C.
5. User A completes Update Set C and migrates it to the production instance.
6. Update Set C commits with warnings.
7. Workflow A is invoked on the production instance with the following results:

   Workflow A fails the runtime validation check and is prevented from running on the production system. The system adds to the workflow context a workflow log entry detailing the cause of the failure, notably the absence of a dependent workflow.

   To learn more about the validation checks on workflow dependencies and update sets see ValidateUpdateSetDependencies.

Use Case - Subflow Dependency (Risk)

Multiple users migrate a workflow from a test instance to a production instance without proper coordination. This use case can succeed, but only when each user understands the dependencies and properly migrates the dependent parts of the workflow to the new instance.

This example does *not* represent an update set failure, although update sets are most often blamed in this type of use case. Validation increases the visibility of workflow dependencies across multiple update sets and provides designers with better information. In most cases, the warnings do not prevent an action but only identify risk. The designer is responsible to follow up on the advice given in the validation checks.

1. User A selects Update Set C.
2. User A checks out Workflow A.
3. User A adds a subflow called Workflow B that returns a **User ID**.

   Assume that Workflow B was previously published and migrated to the production instance.
4. User A uses the return value of Workflow B to generate approvals.
5. User B selects Update Set D.
6. User B checks out Workflow B (the subflow in Workflow A).
7. User B modifies the return value of the workflow by changing it from a **User ID** to a **String Message**.
8. User A publishes Workflow A.

   A dialog box displays warnings associated with Workflow A and encourages User A to validate the workflow before publishing it.

   User A cancels publishing and validates Workflow A.

   User A is warned that Workflow B was modified by a user in a different update set.
9. User A ignores this warning and publishes Workflow A.

   A customer update set record is added to Update Set C containing an XML payload, including the published Workflow A and all activity dependencies.The XML payload also contains the workflow input variables associated with the workflow.
10. User A completes Update Set C and migrates it to the production instance.
11. Workflow A is invoked on the production instance and runs successfully using the older version of Workflow B already on the system.
12. User B publishes Workflow B.

    User B is not warned of the Update Set C dependency, because the update set is no longer **In progress**. However, User B is informed via a dialog box that there are warnings associated with the workflow version and is instructed to validate Workflow B.

    If User B cancels publication and validates the workflow, User B is warned that there are workflows that use Workflow B as a subflow. Knowing the return value was changed, User B should test those workflows as well. See ValidateUpdateSetDependencies to understand the parameters of update set warnings.
13. User B finally publishes Workflow B.

    A customer update set record is added to Update Set D containing an XML payload, including the published Workflow B and all activity dependencies.
14. User B completes Update Set D and migrates it to the production instance.
15. Update Set D commits without warnings.
16. Workflow A is invoked on the production instance and fails to run successfully, because the return value of Workflow B no longer generates a **User ID**.

Moving Input Variables

You can add input variables to existing workflows and add them to update sets.

When you submit the new variables, an entry is made into the current update set that reflects the addition of a variable to the Variables [var_dictionary] table. Unlike the workflow version that only writes to the update set when the workflow is published, the variables write individual update entries into the currently selected update set immediately upon submission.

Use Case

An existing workflow already contains two input variables.

1. User A checks out the workflow.
2. User A adds two input variables.

   ServiceNow adds to the current update set one customer update record for each new variable.

   The current workflow now has 4 input variables: the two that were present prior to check out and the two new ones.
3. User A publishes the workflow.

   There are now three related customer update records: two for new variables, and one for the published workflow. The XML payload of the new workflow version now includes all input variable database entries. So while the two original input variables do not have individual customer update records, all four variables are migrated to the local instance with the payload of the newly published workflow version.
4. To verify the variables that are included in a specific workflow version:
   1. Navigate to **System Update Sets > Local Update Sets**.
   2. Select the active update set.
   3. Select the customer update entry for the workflow version.
   4. View the XML **Payload**.
   5. Search for the name of one of the columns or search for **var_dictionary**.

      There is one **var_dictionary** entry for each input variable.
5. User A completes the update set.

Adding Input Variables - Success

User A migrates and commits the update set to a local instance where the original workflow version had previously been committed.

- The two existing input variables are already present because of the earlier version.
- The system adds the two new input variables when the user commits the update set.
- The system preserves the two legacy input variables on the instance receiving the update set. The update set does not overwrite these variables.
- The new published workflow version uses all four variables.
- The user tests the new workflow version and it runs as expected.

Removing Input Variables

Deleting workflow input variables, like insert and update actions, creates a customer update record in a user's current update set. These deletions migrate to a new instance with the update set, regardless of whether the workflow that owns the input variables is published in the same update set. Plan carefully and use caution when editing a workflow and selecting update sets.

Risk

An existing workflow already contains two input variables.

1. The workflow was migrated to a production instance with the two variables.
2. On a development instance, User A selects Update Set A and checks out the workflow.
3. User A removes one input variable and all references to it in the workflow.

   ServiceNow enters into Update Set A one customer update record reflecting the deletion of the input variable. No record is added for the new workflow version which no longer depends on the input. This does not happen until the workflow is published.
4. User A continues working on other features in Update Set A that need to be moved to production.
5. User A completes Update Set A and migrates it to the production instance without publishing the workflow.

   The update set entry that deletes the workflow input variable now applies to the production instance. The prior version of the workflow is running on this instance and still references the missing variable.

Solution

When editing workflows, particularly when deleting input variables, be sure to use a single update set for all variable editing and workflow publishing. If necessary, merge the update set into a more general set targeted for deployment after the workflow is published.

Warning: If a workflow version is already running on a production system and input variables are deleted from a newer version, those deletions could affect transactions already running against the earlier version. Use extreme caution when deleting workflow input variables and plan the migration carefully.

Prevention

Prior to publishing a workflow version, ServiceNow validates the workflow model to assist the designer in planning for deployment. This validation warns of critical errors that can prevent a workflow from running successfully, but also warns of dependencies and conflicts in update sets. See ValidateUpdateSetDependencies for more details.

Avoiding Duplicate Workflows

Update sets manage the published state of all versions of a workflow prior to committing the workflow version on a local instance. The last version of a workflow committed as an Insert or Update via an update set becomes the currently published version, regardless of the publishing sequence for the workflow versions.

To commit a workflow in an update set:

1. Workflow A - Version 1 is created and published in Update Set A.
2. Update Set A is completed and migrated to a local instance.
3. When the update set is committed, the system sets all prior versions of Workflow A to *published = false*.

   In the first migration, there are no prior versions.
4. Workflow A - Version 1 becomes the only published version of the workflow.

It is not possible to have multiple published versions as a result of update set commits. However, this does not eliminate risk, and care should be taken when migrating update sets. Consider this example:

1. Workflow A - Version 1 is migrated and committed to the production instance.
2. Update Set B is created.
3. Update Set C is created.
4. Workflow A - Version 2 is published in Update Set B.

   A customer update record is added to Update Set B with the Version 2 payload.

   A customer update record is added to Update Set B with the Version 1 workflow left unpublished.
5. Update Set B is completed.
6. Workflow A - Version 3 is published in Update Set C.

   A customer update record is added to Update Set C with the Version 3 payload.

   A customer update record is added to Update Set C with the Version 2 workflow left unpublished.
7. Update Set C is completed.
8. Update Set C is migrated and committed to the production instance.

   Workflow A - Version 1 is set to unpublished.

   Workflow A - Version 2 update is skipped since Update Set B, which contains Version 2, was never migrated.

   Workflow A - Version 3 is committed and becomes the only published version of the workflow.

Risk

Update Set B is migrated and committed to the production instance.

1. Workflow A - Version 3 is set to unpublished.
2. Workflow A - Version 1 remains unpublished.
3. Workflow A - Version 2 is committed and becomes the only published version of the workflow.

   The workflow has gone back a version, perhaps unintentionally. The regressed version becomes the currently published version.

References

[1] https://docs.servicenow.com/bundle/jakarta-servicenow-platform/page/administer/workflow-administration/concept/c_WorkflowMovementWithUpdateSets.html

Article Sources and Contributors

System Update Sets *Source*: http://wiki.servicenow.com/index.php?oldid=251170 *Contributors*: Annmarie, Brozi, CapaJC, Cheryl.dolan, Emily.partridge, Fuji.publishing.user, G.yedwab, Guy.yedwab, Jared.laethem, Jerrod.bennett, Jessi.graves, John.ramos, Joseph.messerschmidt, Michael.hoefer, Peter.smith, Phillip.salzman, Rachel.sienko, Roy.lagemann, Steven.wood, Suzanne.smith, Vaughn.romero, Vhearne

Getting Started *Source*: http://wiki.servicenow.com/index.php?oldid=250602 *Contributors*: Cheryl.dolan, Dawn.bunting, Debbie.bodinger, Emily.partridge, Fuji.publishing.user, G.yedwab, Guy.yedwab, John.ramos, Joseph.messerschmidt, Rachel.sienko, Roy.lagemann, Steven.wood, Vaughn.romero

Administering Update Sets *Source*: http://wiki.servicenow.com/index.php?oldid=250024 *Contributors*: Fuji.publishing.user, John.ramos, Joseph.messerschmidt, Katharine.sohler, Publishing.user, Rachel.sienko, Roy.lagemann, Vaughn.romero

Using Update Sets *Source*: http://wiki.servicenow.com/index.php?oldid=251036 *Contributors*: Anat.kerry, Annmarie, CapaJC, Cesar.sandoval, Cheryl.dolan, Danijel.stanojevic, David Loo, Don.Goodliffe, Fuji.publishing.user, G.yedwab, Gadi.yedwab, Guy.yedwab, John.ramos, John.roberts, Joseph.messerschmidt, Michael.hoefer, Pat.Casey, Rachel.sienko, Roy.lagemann, Steven.wood, Vaughn.romero, Vhearne, Wallymarx

Transferring Update Sets *Source*: http://wiki.servicenow.com/index.php?oldid=251080 *Contributors*: Amy.bowman, Cheryl.dolan, Emily.partridge, Fuji.publishing.user, John.ramos, Joseph.messerschmidt, Michael.hoefer, Phillip.salzman, Publishing.user, Rachel.sienko, Roy.lagemann, Vaughn.romero

Saving Customizations in a Single XML File *Source*: http://wiki.servicenow.com/index.php?oldid=251164 *Contributors*: Cheryl.dolan, G.yedwab, Guy.yedwab, John.ramos, Joseph.messerschmidt, Peter.smith, Rachel.sienko, Steven.wood, Vaughn.romero

Moving Workflows with Update Sets *Source*: http://wiki.servicenow.com/index.php?oldid=250332 *Contributors*: Cheryl.dolan, Debbie.bodinger, Frankie.thompson, Fuji.publishing.user, G.yedwab, Ishrath.razvi, Jennifer.ball, Joe.Westrich, John.ramos, Joseph.messerschmidt, Julie.phaviseth, Michael.hoefer, Rachel.sienko, Steven.wood

Image Sources, Licenses and Contributors

Image:Warning.gif *Source*: http://wiki.servicenow.com/index.php?title=File:Warning.gif *License*: unknown *Contributors*: CapaJC

Image:Caution-diamond.png *Source*: http://wiki.servicenow.com/index.php?title=File:Caution-diamond.png *License*: unknown *Contributors*: John.roberts, Publishing.user

Image:updatesetapp.png *Source*: http://wiki.servicenow.com/index.php?title=File:Updatesetapp.png *License*: unknown *Contributors*: Fuji.publishing.user, G.yedwab

Image:Update set picker.png *Source*: http://wiki.servicenow.com/index.php?title=File:Update_set_picker.png *License*: unknown *Contributors*: Fuji.publishing.user, G.yedwab

Image:Us3.png *Source*: http://wiki.servicenow.com/index.php?title=File:Us3.png *License*: unknown *Contributors*: CapaJC, Don.Goodliffe, Fuji.publishing.user

Image:UpdateSetRecord.png *Source*: http://wiki.servicenow.com/index.php?title=File:UpdateSetRecord.png *License*: unknown *Contributors*: Fuji.publishing.user, Rachel.sienko

Image:ShowRecord.png *Source*: http://wiki.servicenow.com/index.php?title=File:ShowRecord.png *License*: unknown *Contributors*: Rachel.sienko

Image:ShowUpdate.png *Source*: http://wiki.servicenow.com/index.php?title=File:ShowUpdate.png *License*: unknown *Contributors*: Rachel.sienko

Image:MergeUpdateSets.png *Source*: http://wiki.servicenow.com/index.php?title=File:MergeUpdateSets.png *License*: unknown *Contributors*: Rachel.sienko

Image:Update_Sets_Collision_Report1.png *Source*: http://wiki.servicenow.com/index.php?title=File:Update_Sets_Collision_Report1.png *License*: unknown *Contributors*: Steven.wood

Image:Update Set Collisions.png *Source*: http://wiki.servicenow.com/index.php?title=File:Update_Set_Collisions.png *License*: unknown *Contributors*: Steven.wood

Image:Update Record.gif *Source*: http://wiki.servicenow.com/index.php?title=File:Update_Record.gif *License*: unknown *Contributors*: Steven.wood

Image:Us20.png *Source*: http://wiki.servicenow.com/index.php?title=File:Us20.png *License*: unknown *Contributors*: CapaJC, Don.Goodliffe

Image:RemoteInstance.png *Source*: http://wiki.servicenow.com/index.php?title=File:RemoteInstance.png *License*: unknown *Contributors*: Rachel.sienko

Image:UpdateSetPreviewProblems.png *Source*: http://wiki.servicenow.com/index.php?title=File:UpdateSetPreviewProblems.png *License*: unknown *Contributors*: Rachel.sienko, Roy.lagemann

Image:PreviewRecord.png *Source*: http://wiki.servicenow.com/index.php?title=File:PreviewRecord.png *License*: unknown *Contributors*: Rachel.sienko

Image:Update_Set_Commit_Confirm.png *Source*: http://wiki.servicenow.com/index.php?title=File:Update_Set_Commit_Confirm.png *License*: unknown *Contributors*: Rachel.sienko, Steven.wood

Image:Saving.png *Source*: http://wiki.servicenow.com/index.php?title=File:Saving.png *License*: unknown *Contributors*: Guy.yedwab

Image:Saving2.png *Source*: http://wiki.servicenow.com/index.php?title=File:Saving2.png *License*: unknown *Contributors*: Guy.yedwab