

# Team Development

# Team Development



**Note:** This article applies to Fuji. For more current information, see *Team Development*<sup>[1]</sup> 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 The ServiceNow Wiki is no longer being updated.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

Team development is a version control system similar to Git. It supports parallel development on multiple, sub-production ServiceNow instances by providing:

- Branching operations, including pushing and pulling record versions between instances.
- The ability to compare a development instance to other development instances.
- A central dashboard for all team development activities.

This feature is active by default starting with the Dublin release.

## When to Use Team Development and When to Use Another Deployment Option

Deployment option	Good for	Future considerations
Team Development	<ul style="list-style-type: none"><li>• Providing change management across multiple instances</li><li>• Allowing multiple developers to work on applications</li><li>• Organizations that have access to several sub-production instances</li></ul>	<ul style="list-style-type: none"><li>• Works best when each development team has access to a dedicated development instance.</li><li>• Requires developers to manually merge colliding changes.</li><li>• Works only for instances owned by the same organization.</li></ul> <div><p><b>Note:</b> If used in conjunction with the application repository, make applications available from a parent instance.</p></div>
Update Sets	<ul style="list-style-type: none"><li>• Storing changes to a baseline or installed application.</li><li>• Storing and applying a particular version of an application.</li><li>• Producing a file for export.</li></ul>	<ul style="list-style-type: none"><li>• You can manually create update sets to store a particular application version.</li><li>• Use update sets to deploy patches or changes to installed applications.</li></ul> <div><p><b>Note:</b> Avoid using update sets to install applications. Instead, use the application repository or the ServiceNow Store to install applications.</p></div>
Application Repository	<ul style="list-style-type: none"><li>• Installing and updating applications on all of your instances</li><li>• Automatically managing application update sets</li><li>• Restricting access to applications to the same company</li><li>• Deploying completed applications to end users</li></ul>	<ul style="list-style-type: none"><li>• Consider uploading an application to the ServiceNow Store to share it with other users.</li><li>• Allows installation of and update to the latest application version only. Use update sets to store prior application versions.</li></ul> <div><p><b>Note:</b> If used in conjunction with team development, make applications available only from a parent instance.</p></div>

# Team Development Concepts

## Instance Hierarchies

Team development allows you to set up a distributed version control system between two or more ServiceNow instances where each instance acts as a source repository, or branch. Developers use separate instances to work on different features, applications, or product releases at the same time. With team development, developers can share code between these instances and resolve collisions throughout the development process.

Team development allows you to establish hierarchical relationships between instances and provides a mechanism for transferring changes between instances that integrates with the update set process where necessary. In a team development instance hierarchy, each sub-production instance has a parent instance. Instances that have the same parent instance are peer instances. The shared parent instance becomes the central hub, or repository, and all peer instances synchronize to it.

## Pulls and Pushes

Developers synchronize their instances to the parent instance by pulling and pushing versions of customized records and resolving collisions between versions on the parent instance and the development instance. Developers can also compare peer instances to one another and share code or resolve collisions before pushing versions to the parent instance.

Pulling from the parent retrieves versions of records that have customer updates. Pulling retrieves all versions that have not already been pulled onto the development instance, including historical versions, and you cannot choose which versions to pull. You must resolve any collisions before proceeding with further pulls or pushes.

Pushing to the parent adds only the current development version to the parent, not all the development versions. You can choose which changes to push to the parent. Pushing creates a local update set on the parent that is marked as complete. Pushed versions are also tracked as local changes on the parent. Therefore, you can promote changes through your development and test hierarchy by transferring the update set or by pushing the local changes.

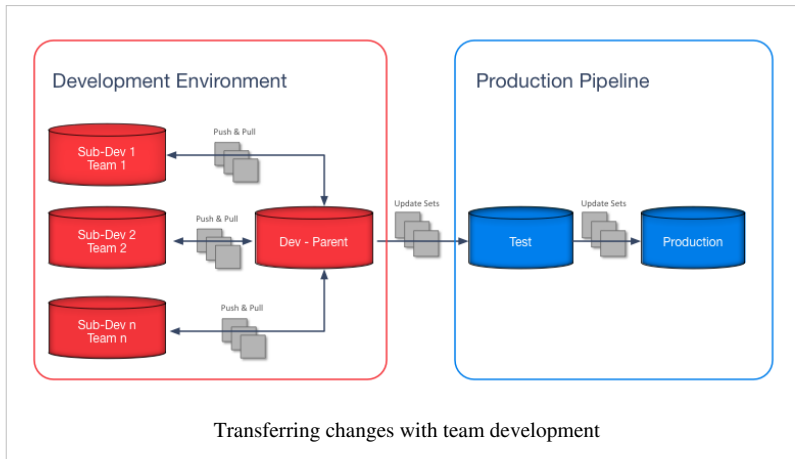
Comparing reports the differences between two peer instances. You can choose which versions to pull from a peer instance.

## Versions and Local Changes

Version records track changes to a customizable record over time so that you can compare or revert to a specific version later. A version record is created every time a developer changes a customizable record, so a single record can have multiple version records associated with it.

Local change records track which customized records have changes on the development instance that are not on the parent instance so that you can collect changes in preparation for a push. A local change record is created or updated to reference the current version every time a developer modifies a customizable record, so a single record can have only one local change record associated with it.

---



## Back Out Local Changes

Developers can back out a local change to restore a previous version of a customizable record. The back out action sets the local customizable record to the last revision identified by a reconciliation action. See [Backing Out Local Changes](#). This feature is available starting with the Eureka release.

## Change Tracking

The **Pushes and Pulls** related list on the team dashboard displays the user who created a change and the remote instance where the change was created, available starting with the Eureka release. In versions prior to Eureka, the team dashboard displays all changes as coming from the system user. The change list does not include the remote instance where the change came from.

## Exclusion Policies

You can exclude certain files from change tracking by creating an exclusion policy. When a change matches an exclusion policy, the change does not generate records in the local changes list. The change still generates local version records and update set records as normal. See [Creating an Exclusion Policy](#). Exclusion policies are available starting with the Eureka release.



**Note:** The exclusion policy applies to changes identified during a reconciliation operation. If you create an exclusion policy after a reconciliation, team development still tracks the changes until the next reconciliation.

## Code Review

Team development administrators can require that pushes undergo code review before accepting them. When code review is enabled, pushing a change to the parent instance triggers the code review workflow. By default, users with the `teamdev_code_reviewer` role receive notifications to review changes and can approve or reject changes. The Team Development Code Reviewers group has the `teamdev_code_reviewer` role.

For each change, reviewers can see the following information.

- What remote instance the pushed change comes from
- Who pushed the change to the parent.
- What the change is called.
- When the change was created.
- What versions the change includes.

Reviewers must approve or reject a push from the Team Development application.

While changes are being reviewed on the parent instance, a child instance cannot do the following activities involving the parent instance:

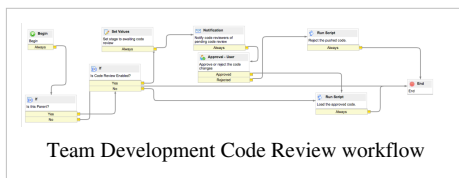
- Push changes to the parent instance.
- Pull changes from the parent instance.
- Reconcile changes with the parent instance.
- Change the parent instance to another instance.
- Delete the parent instance's remote instance record.


The option to require code review is available starting with the Eureka release.

## Code Review Workflow

The Team Development Code Review workflow manages how changes are pushed to the parent. By default this workflow:

- Starts when changes are pushed to the parent instance.
- Verifies the code review property is active on the parent instance.
- Sets the stage of changes requiring approval to **Awaiting Code Review**.
- Notifies the Team Development Code Reviewers group to review pushed changes, if configured.
- Loads approved changes or sets the stage to **Code Changes Rejected**.



	<p><b>Warning:</b> Use caution when modifying this workflow, as the code review feature may not function properly.</p>
---	--

## Notifications

You must enable email notifications on the instance requiring code review for that instance to send code review notifications. See [Configuring Email](#). The Team Development Code Review workflow sends notifications to members of the Team Development Code Reviewers group when:

- A push requires code review.
- A user cancels a push.

If the user who pushed the changes has a user record with an email address on the instance where code review was required, the user receives a notification when the approval stage is set to **Complete** (approved) or **Code Changes Rejected**.

The code review notifications contain the following information:

Notification Name	Table	Contents
Code review update for developer	Push or Pull [sys_sync_history]	<ul style="list-style-type: none"> <li>The push name</li> <li>The approval stage of the push (approved or rejected)</li> <li>A link to the instance where the code review request was made</li> </ul>
Notify code reviewer of cancelled review	Push or Pull [sys_sync_history]	<ul style="list-style-type: none"> <li>The user who cancelled review</li> <li>The push that was cancelled</li> </ul>

## Team Development Process



**Warning:** Do not use Team Development with production or test instances.

- Do not use a test or production instance as the parent instance in Team Development.
- Do not make any instance the parent of a production instance.
- Production instances should never have a parent.

When you back out a change on a Team Development instance, it backs out the change all the way back down the chain, including undoing the work on the source instance. This behavior can cause major problems on test and production instances.

Follow this basic process to use team development:

- Set up the development instance hierarchy.
  - Provision development instances on the same software version as the target instance. For example, use the software version that is running on your production instance.
  - [Recommended] Clone the target to the development instances.
  - For each instance, define the parent instance.
  - [Optional] For each instance, define the peer instances.
  - For each instance, pull all changes from the parent instance.
- For sub-development instances, grant access rights to appropriate developers.
- Develop customizations on sub-development instances. Use the team dashboard to track development activities.
  - Pull versions from the parent instance, such as versions that were pushed from other sub-development instances. Reconcile any conflicts with the current local version, as necessary.
  - Track local changes. Queue changes that are ready to push to the parent development instance.
  - Compare versions on peer instances. Reconcile any conflicts.
- When a feature is ready to promote to the parent development instance, push the current version of the customized records.
- [Optional] Have code reviewers approve or reject the pushed version.
- Test and promote the feature into production according to your testing and release management process.

## Roles

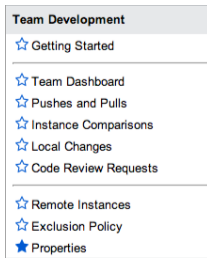
To use team development, developers must have admin access to their development instance. To allow pushes to the parent instance, a remote instance connection must be defined with a user account that has admin access to the parent instance.

To limit developer access to the parent instance, see [Granting Access Rights to Developers](#).

To use code review features, users must have the `teamdev_code_reviewer` role. See [Code Review](#).

## Menus and Modules

To access team development features, use these modules under the **Team Development** menu.



- **Team Dashboard:** central dashboard for all team development activities. See Team Dashboard.
- **Pushes and Pulls:** history of pushes and pulls between the local instance and the parent instance. See Pulling Versions and Pushing Versions.
- **Instance Comparisons:** history of comparisons between the local instance and peer instances. See Comparing to Peer Instances.
- **Local Changes:** list of customized records on this instance that are not present on the parent instance. See Local Changes.
- **Code Review Requests:** list of changes pushed to this instance requiring code review. See Code Review.
- **Remote Instances:** list of other instances in the hierarchy for which connection settings are defined. See Defining Remote Instances.
- **Exclusion Policy:** list of changes that do not generate records in the local changes list. See Exclusion Policies.
- **Properties:** list of team development options. See Setting Up Team Development.

## Enhancements

### Fuji

- Developers can back out completed pushes, committed remote update sets, and completed local update sets.
- The local changes list shows only the latest version for a workflow.

### Eureka

- Allows team developers to back out local changes they do not want to keep.
- Allows team development administrators to view change tracking information about the user who submitted a change and the remote instance it came from.
- Allows team development administrators to create exclusion policies to prevent an instance from tracking changes to particular records.
- Allows team development administrators to require code review of all changes pushed to an instance.
- Allows team development administrators to specify whether a remote instance is active.
- Requires a remote instance to be on the same major version in order to be selected as a parent instance.
- Allows team development operations to be completed by non-administrators.
- Allows developers to receive notifications for team development events.

## References

- [1] [https://docs.servicenow.com/bundle/jakarta-application-development/page/build/team-development/concept/c\\_TeamDevelopment.html](https://docs.servicenow.com/bundle/jakarta-application-development/page/build/team-development/concept/c_TeamDevelopment.html)

# Setting Up Team Development

## Overview

To enable parallel development on multiple sub-production instances, administrators can set up the team development instance hierarchy and grant access rights for developers.

## Setting Up an Instance Hierarchy

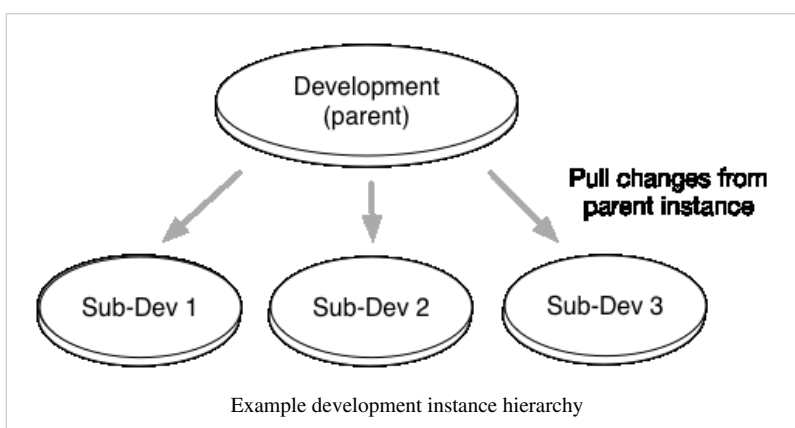
Set up an instance hierarchy that best supports your development life cycle.



**Note:** Production, test, and user acceptance test (UAT) instances should not be configured to use Team Development. Your Development system should not use Team Development to move changes into test.

The following example demonstrates how to set up an instance hierarchy where several peer sub-development instances have the same parent development instance, but a more complex configuration may be required to handle multiple project teams or other customer requirements.

1. Provision a parent development instance on the same software version, such as Dublin, as the target instance, such as production.
2. [Recommended] Clone the production instance to the parent development instance. See System Clone.
3. Provision sub-development instances on the same software version as the parent development instance.
4. [Optional] Log in to the parent development instance and clone it to the sub-development instances.
5. On each sub-development instance:
  1. Define remote instance connections to other instances in the hierarchy that this instance needs to push and pull with.
  2. Select the parent instance.
  3. Pull all changes from the parent instance. See Pulling Versions.
  4. Grant access rights to appropriate developers.



## Defining Remote Instances

For each instance, define other instances in the hierarchy as remote instances. For example, to set up remote instances for Sub-Dev 1:

1. If IP address access control is enabled, log in to the remote instance and add Sub-Dev 1 as an exception.
2. On Sub-Dev 1, navigate to **Team Development > Remote Instances**.
3. Click **New**.
4. Define the remote instance, such as Dev-Parent, by completing the form (see table).



Remote Instance

Required field

UpdateDeleteTest Connection

Name:Dev - ParentURL:https://demo.service-now.com

Type:DevelopmentUsername:admin

Password:\*\*\*\*\*

Short description:Development instance

UpdateDeleteTest Connection

Related Links

[Retrieve Completed Update Sets](#)

[Compare to Local Instance](#)

[Make This Your Parent](#)

Define the remote instance

5. Click **Submit**.
6. Repeat steps 1–5 for each instance in the hierarchy that this instance needs to push and pull with (for example, Sub-Dev 2 and Sub-Dev 3).

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 communicates with the remote instance as a member of team development. Team development operations such as comparing changes between instances or selecting a parent instance are only available for 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.
Username	Enter the user on the remote instance who authorizes Team Development operations on the instance. This user account must have an appropriate role on the remote instance. See Granting Access Rights to Developers.
Password	Enter the password of the authorizing user.
Short description	[Optional] Enter any other relevant information about the remote instance.

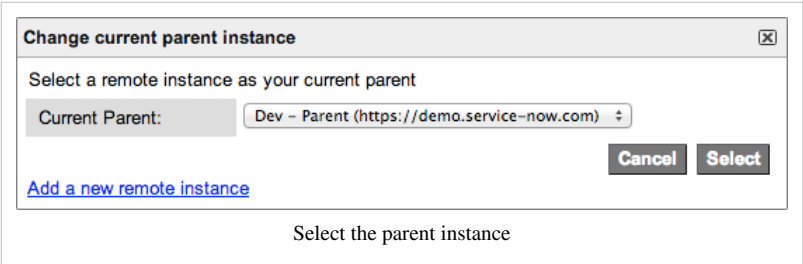
Selecting the Parent Instance

An instance can have multiple peer instances but only one parent instance. The parent instance is the only instance you can pull changes from and push changes to.

The parent instance must be on the same release family as the local instance (starting with the Eureka release). For example, a development instance on the Eureka release family must have a parent instance also on the Eureka release family. If you select a parent from a different release family, the team development dashboard displays an error message and prevents you from pulling changes and reconciling. If you select a parent from a different patch release, the dashboard displays a warning message but allows you to pull changes and reconcile.

To select the parent instance:

1. Navigate to **Team Development > Team Dashboard**.
2. In the control panel, click the appropriate link:
  - **Use <instance name and URL>:** selects the most recently defined remote instance as the parent instance.
  - **Select a different instance:** opens a dialog box where you can select another remote instance or define a new remote instance.
  - **Register a new instance** or **List all remote instances:** opens the remote instance form or list, where you can define a new remote instance. These options are available when no remote instances are defined.



- generates the list of local changes and calculates the number of changes that are ready to pull from the parent. The reconcile also validates the instance versions.
4. Pull all changes from the parent instance if both instances are in the same release family (starting with the Eureka release). See Pulling Versions.



**Note:** The parent instance is saved in the `glide.apps.hub.current` system property.

## Granting Access Rights to Developers

To use team development, developers must have a set of credentials for each instance in the team development hierarchy. The following credentials are required for the Eureka release. If you are using an older version of ServiceNow, see the previous version information.

Desired Access	Credential Requirements
Access to the Team Development application	A user with the admin role on the instance you are accessing
Right to register a remote instance	One of following: <ul style="list-style-type: none"><li>• A user with the admin role on the instance you are registering</li><li>• A user with the teamdev_user role on the instance you are registering (starting with the Eureka release)</li></ul>
Right to push changes to the parent instance from a development instance	One of following: <ul style="list-style-type: none"><li>• A user with the admin role on the parent instance</li><li>• A user with the teamdev_user role on the parent instance (starting with the Eureka release)</li></ul>
Right to compare to a registered remote instance	One of following: <ul style="list-style-type: none"><li>• A user with the admin role on the registered development instance</li><li>• A user with the teamdev_user role on the registered development instance (starting with the Eureka release)</li></ul>
Access to the Code Review Requests module	One of following: <ul style="list-style-type: none"><li>• A user with the admin role on the parent instance</li><li>• A user with the teamdev_code_reviewer role on the parent instance (starting with the Eureka release)</li></ul>



**Note:** The `teamdev_user` role does not grant access to the Team Development application and is not intended for developers to work on local development instances. It is intended to grant developers non-admin access to remote instances such as the parent instance or a peer development instance.

*Click the plus for previous version information*

Desired Access	Credential Requirements
Access to the Team Development application	A user with the admin role on the instance you are accessing
Right to register a remote instance	A user with the admin role on the instance you are registering †
Right to push changes to the parent instance from a development instance	A user with the admin role on the parent instance †
Right to compare to a registered remote instance	A user with the admin role on the registered development instance

† If you need to restrict admin access to a remote instance to prevent developers from making changes on that instance, ServiceNow recommends using update sets instead of Team Development. The modified process for developing and promoting changes is:

1. Developers use update sets to collect changes on their development instance.
2. When the feature is complete, developers complete the update set on their development instance and inform an administrator for the parent instance.
3. The administrator for the parent retrieves and commits the update set according to the established testing and release management process.



**Note:** Using update sets, developers cannot pull changes nor reconcile with changes from a parent instance.

## Creating an Exclusion Policy

To create an exclusion policy:

1. Navigate to **Team Development > Exclusion Policy**.
2. Click **New**.
3. Complete the Exclusion Policy form (see table).
4. Click **Submit**.

Field	Description
<b>Name</b>	Enter a unique description of the policy.
<b>Policy</b>	Select when the policy applies. Options include: <ul style="list-style-type: none"> <li>• Push only</li> <li>• Push and Pull</li> <li>• Pull only</li> </ul>
<b>Remote Instance</b>	[Optional] Select a specific remote instance to ignore changes from during pull operations. Leaving this field blank ignores changes from <i>all</i> remote instances.
<b>Table</b>	Select which table to ignore changes for.
<b>Conditions</b>	Select any additional criteria a change must meet to be ignored other than the table name. This field is only visible when the <b>Policy</b> is <i>Push only</i> .

## Enabling Code Review

To require code review of all changes pushed to an instance:

1. Navigate to **Team Development > Properties**.
2. Select the **Yes** check box for **If this property is set to Yes, code review is required before pushing to this instance** (`com.snc.teamdev.requires_codereview`).
3. Click **Save**.

Setting this property adds the Code Review Requests module to the application menu and requires all changes pushed to this instance to remain in the **Awaiting Code Review** stage until someone in the Team Development Code Reviewers group approves them.

## Using Team Development

---

### Overview

Team development allows developers to work on separate development instances while sharing code and resolving collisions throughout the development process. Be sure to read Team Development before starting to use team development. After setting up the instance hierarchy, develop changes on your local development instance. Use the team dashboard to manage team development activities, such as:

- Tracking local changes and determining which changes to promote to the parent development instance.
- Pulling changes from the parent instance and resolving any collisions with local changes.
- Comparing your instance with other development instances and resolving any collisions with other development projects.
- Pushing changes when a feature is tested and ready to promote to the parent development instance.

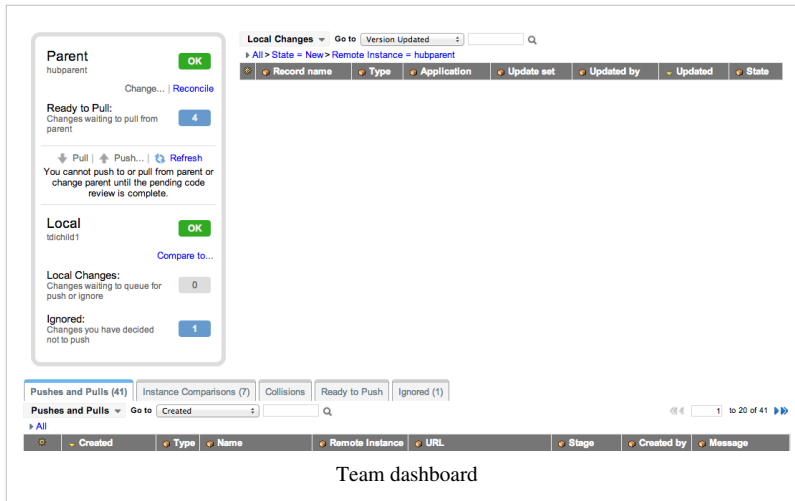
Developers with admin access to their development instance and the parent instance can use team development. For alternative access settings, see Granting Access Rights to Developers.

### Team Dashboard

The team dashboard provides a central place to manage all team development activities on your development instance. You can track local changes, pull and push changes between the local and parent instances, compare the local instance to other development instances, and resolve any collisions. You can also reconcile with the current parent instance or change the parent instance.

To access the dashboard, navigate to **Team Development > Team Dashboard**.

---



The control panel in the top left provides status indicators and team development actions.

- **Parent:** indicates the status of the connection to the parent instance. If a problem or warning is detected, point to the indicator to view the error messages, or click the indicator to open the remote instance record.
- **Change:** changes the parent instance. See Changing the Parent Instance.

- **Reconcile:** compares the development instance to the parent instance. See Reconciling.
- **Ready to Pull:** indicates the number of changes on the parent that have not been pulled to the local instance.
- **Pull:** initiates a pull. See Pulling Versions.
- **Push:** opens a page that allows you to review the changes before a push. See Pushing Versions.
- **Refresh:** updates the status indicators on the control panel. The dashboard updates only when you reload or refresh the page.
- **Local:** indicates the status of the most recent comparison with another instance. If collisions are detected, click the indicator to open the list and resolve the collisions. See Resolving Collisions.
- **Collisions:** appears only if any local changes collide with versions pulled from the parent and indicates the number of collisions. Click the indicator to open the list and resolve the collisions.
- **Compare to:** allows you to select another development instance to compare with the local instance. See Comparing to Peer Instances.
- **Ready to Push:** indicates the number of local changes that are queued for the next push. See Queuing and Ignoring Local Changes.
- **Local changes:** indicates the number of local changes that have not been queued or ignored. Click the indicator to open a list of these changes.
- **Ignored:** appears only if any local changes are ignored and indicates the number of ignored changes. Click the indicator to open a list of these changes.

The team dashboard includes lists for tracking local changes and viewing the history of team development activities.

- **Local changes:** lists the local changes that have not been queued or ignored.
- **Pushes and Pulls:** provides a history of pushes and pulls. Expand a row to see the customized records for which versions were transferred as part of the push or pull.
- **Instance Comparisons:** provides a history of comparisons with other development instances.
- **Collisions:** lists the collisions that must be resolved before the next pull or push. You can right-click a row and select **Resolve Collision**. See Resolving Collisions.
- **Ready to Push:** lists the local changes that have been queued for the next push.
- **Ignored:** lists the local changes that are ignored for all pushes.

Pulling retrieves versions of customized records from the parent instance and adds them on the development instance. Pulling does not retrieve any versions for changes made by system upgrades, but it retrieves all versions for changes made by users, not just the current version. Historical versions are saved with a state of **History**.

To pull versions from the parent instance:

- The **Push and Pull Versions** related list shows the customized records for which versions were retrieved and indicates if any pull exceptions exist.

4. Resolve any collisions.

Pulling ignores versions when any of the following conditions occur:

Issue	Description
Matched an exclusion policy	An exclusion policy prevents pulling changes for records matching the policy conditions. The pull identifies the changes but does not include versions for these records. Exclusion policies are available starting with the Eureka release.
Private properties	A private property is excluded from all update sets and pulls.
Collisions	A collision is detected when the pulled version and the current local version both include modifications to the same record. You must resolve all collisions before you can pull.
Previously resolved collisions	A previously resolved collision is when you resolved a collision by accepting either the pulled version or local version of a record. The pull remembers your decision and accepts the version you indicated.
Skipped	Pulls will skip versions where there is a problem with the version record such as a corrupt or missing version.

## Resolving Collisions

The team dashboard displays the number of collisions between the local and the parent instance. A collision is detected when the pulled version and the current local version are modifications of a different version, indicating that someone else has modified the same record you have modified. To ensure that your changes do not conflict with other development efforts, you should resolve collisions as soon as they are identified. You must resolve all collisions before you can pull or push.

To resolve collisions individually:

1. Navigate to **Team Development > Team Dashboard**.
2. In the control panel, click the number of collisions. A list of collisions opens.
3. Right-click a row and select **Resolve Collision**. A comparison between the current version on the local instance and the current version pulled from the parent appears.
4. Review the differences between the versions and use one of the following methods to resolve the collision:
  - Click **Use pulled version** to load the version pulled from the parent as the current version.
  - Click **Use local version** to maintain the local version as the current version. The pulled version is added to the version history for the record.
  - Create a new local version and manually merge the changes in both the pulled and local versions. Then, repeat steps 1–4 for the same change and click **Use local version**.

	Pulled version	Local version
fields 1:	u_refer_table_a	1: u_refer_table_a
2:	sys_updated_on	2: sys_created_on
3:	sys_updated_by	3: sys_created_by

Select the local or pulled version

5. In the confirmation dialog box, click **OK**.
6. Repeat steps 3–6 for every collision in the list.

To resolve multiple collisions without reviewing the differences between the local and pulled versions:

1. Navigate to **Team Development > Team Dashboard**.
2. In the control panel, click the number of collisions. A list of collisions opens.
3. Select the check boxes beside the rows you want to resolve.
4. In the **Actions** choice list, use one of the following methods to resolve the collision:
  - Select **Use Pulled Version** to load the version pulled from the parent as the current version for all selected collisions.
  - Select **Use Local Version** to maintain the local version as the current version for all selected collisions. The pulled versions are added to the version history for the records.

## Local Changes

The Local Changes table tracks which customized records have current versions that exist on the development instance but not on the parent instance. Use local changes to collect changes in preparation for a push.

You *queue* local changes that are ready to push. Each development instance maintains a single queue, regardless of who develops or queues the changes. You *ignore* local changes that you do not want to push. For example, you may want to ignore changes to the color scheme that visually distinguish a development instance from the production instance. You can remove a change from the queue or stop ignoring a change.

Changing the parent instance or reconciling recreates the list of local changes that have not been queued or ignored. If you had previously queued or ignored a local change, that designation is maintained.

## Navigating Local Change Lists

On the team dashboard, the **Local Changes** list shows the local changes that have not been queued for the next push or ignored for all pushes. The **Ready to Push** list shows the changes that are queued, and the **Ignored** list shows the changes that are ignored. Use any of these methods to navigate a list of local changes.

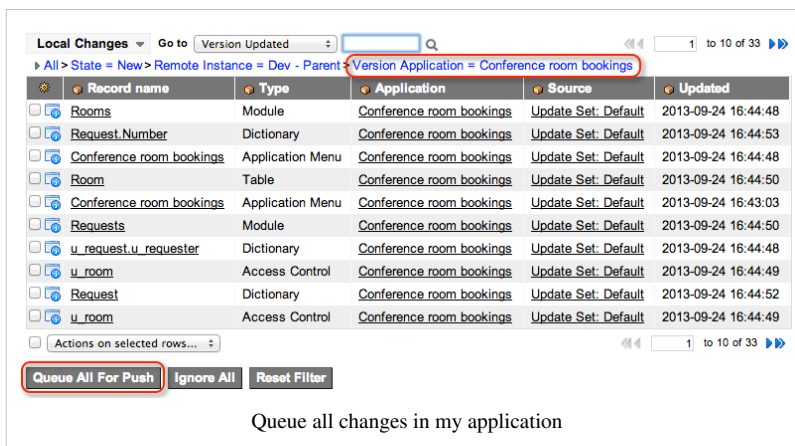
- To open the local change record itself, click the reference icon beside the row.
- To open the customized record, click the link in the first column.
- To view a comparison between the current local version and the version most recently pulled from or pushed to the parent, right-click the row and select **Show Changes Since Last Pull**. An error message appears if a previous version does not exist (for example, in the case of a newly created record). If a previous version is available, you can revert to that version from the comparison window.
- To open the application file for the customized record, right-click the row and select **Show Application File**.
- To open the current version record, right-click the row and select **Show Version**.

## Queuing Local Changes for Push

To queue changes that are ready to push:

1. Navigate to **Team Development > Team Dashboard**.
2. Filter the **Local Changes** list to show only the changes that are ready to push.

For example, filter the list to show only the changes associated with a particular application.



Local Changes

Go to: Version Updated

Version Application = Conference room bookings

Record name	Type	Application	Source	Updated
Rooms	Module	Conference room bookings	Update Set: Default	2013-09-24 16:44:48
Request Number	Dictionary	Conference room bookings	Update Set: Default	2013-09-24 16:44:53
Conference room bookings	Application Menu	Conference room bookings	Update Set: Default	2013-09-24 16:44:48
Room	Table	Conference room bookings	Update Set: Default	2013-09-24 16:44:50
Conference room bookings	Application Menu	Conference room bookings	Update Set: Default	2013-09-24 16:43:03
Requests	Module	Conference room bookings	Update Set: Default	2013-09-24 16:44:50
u_request.u_requester	Dictionary	Conference room bookings	Update Set: Default	2013-09-24 16:44:48
u_room	Access Control	Conference room bookings	Update Set: Default	2013-09-24 16:44:49
Request	Dictionary	Conference room bookings	Update Set: Default	2013-09-24 16:44:52
u_room	Access Control	Conference room bookings	Update Set: Default	2013-09-24 16:44:49

Queue All For Push Ignore All Reset Filter

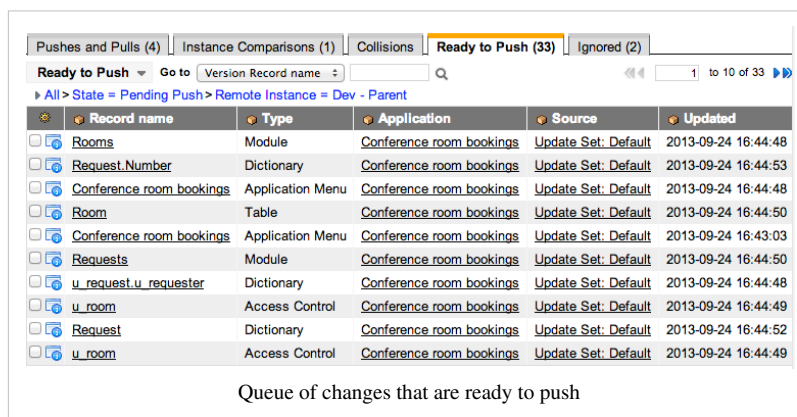
Queue all changes in my application

3. Click **Queue All For Push**.

4. [Recommended] Review the **Ready to Push** list to ensure that the correct changes are in the queue.

- To remove changes from the queue, select the check boxes beside the rows and select **Do Not Push** from the **Actions** choice list.
- To remove changes from the queue and choose to ignore them

instead, select the check boxes beside the rows and select **Ignore This Change** from the **Actions** choice list.



Pushes and Pulls (4) Instance Comparisons (1) Collisions

Ready to Push (33) Ignored (2)

Ready to Push

Go to: Version Record name

State = Pending Push Remote Instance = Dev - Parent

Record name	Type	Application	Source	Updated
Rooms	Module	Conference room bookings	Update Set: Default	2013-09-24 16:44:48
Request Number	Dictionary	Conference room bookings	Update Set: Default	2013-09-24 16:44:53
Conference room bookings	Application Menu	Conference room bookings	Update Set: Default	2013-09-24 16:44:48
Room	Table	Conference room bookings	Update Set: Default	2013-09-24 16:44:50
Conference room bookings	Application Menu	Conference room bookings	Update Set: Default	2013-09-24 16:43:03
Requests	Module	Conference room bookings	Update Set: Default	2013-09-24 16:44:50
u_request.u_requester	Dictionary	Conference room bookings	Update Set: Default	2013-09-24 16:44:48
u_room	Access Control	Conference room bookings	Update Set: Default	2013-09-24 16:44:49
Request	Dictionary	Conference room bookings	Update Set: Default	2013-09-24 16:44:52
u_room	Access Control	Conference room bookings	Update Set: Default	2013-09-24 16:44:49

Queue of changes that are ready to push



**Note:** For the **Local Changes** list, click **Reset Filter** to remove any filter conditions you added and see all the local changes that have not been queued or ignored.



## Ignoring Local Changes

Ignoring a local change prevents updates to a record from generating new versions in the Local Changes list. An ignored local change always points to the current version for the record. You cannot push ignored records to another instance.

Action	Results
Ignore a record that has a version queued for push	The queued change is deleted
Ignore a record that has a version queued for code review	The queued change is deleted
Pull changes for an ignored record	Collision
Resolve a collision by taking the parent version	There is no longer a local change to ignore
Resolve a collision by keeping the local version	The ignored change remains on the local instance

To ignore changes that you do not want to push:

1. Navigate to **Team Development > Team Dashboard**.
2. Filter the **Local Changes** list to show only the changes that you want to ignore.

For example, filter the list to show all changes in the **Default** update set.

Local Changes

Go to: Version Application [ ] Q

1 to 10 of 11

All > State = New > Remote Instance = Dev - Parent > Version Source = 88eacc457c00110026c11c2fdf262834

Record name	Type	Application	Source	Updated
Table B	Application Menu		Update Set: Default	2013-09-24 16:28:31
Hardware	Category		Update Set: Default	2013-09-24 14:24:32
Default1	Homepage Category Renderer		Update Set: Default	2013-09-24 13:38:46
Sys_popup.item	UI View		Update Set: Default	2013-09-24 13:34:30
sc_catalog_homepage_category_list1	Macro		Update Set: Default	2013-09-24 13:38:52
Document Services	Category		Update Set: Default	2013-09-24 13:34:38
Default	Homepage Category Renderer	Service Catalog	Update Set: Default	2013-09-24 14:24:39
Category Only	Homepage Category Renderer	Service Catalog	Update Set: Default	2013-09-24 13:40:24
com.snc.sla.engine.version	System Property	Service level management	Update Set: Default	2013-09-24 03:29:09
SLA Condition Rules	Module	Service level management	Update Set: Default	2013-09-24 03:29:09

Actions on selected rows...

Queue All For Push Ignore All Reset Filter

Ignore all changes in the Default update set

3. Click **Ignore All**.
4. [Recommended] Review the **Ignored** list to ensure that the correct changes are ignored.
  - To stop ignoring changes, select the check boxes beside the rows and select **Do Not Ignore** from the **Actions** choice list.
  - To stop ignoring changes and add them to the queue instead, select the check boxes beside the rows and select **Queue for Push** from the **Actions** choice list.

Pushes and Pulls (4) Instance Comparisons (1) Collisions Ready to Push (33) Ignored (13)

Ignored Go to: Version Application [ ] Q

1 to 10

All > State = Ignored > Remote Instance = Dev - Parent

Record name	Type	Application	Source	Updated
Test relative duration	SLA		Update Set: Default	2013-09-24
End Next B Day	Relative Duration		Update Set: Default	2013-09-24
Document Services	Category		Update Set: Default	2013-09-24
Sys_popup.item	UI View		Update Set: Default	2013-09-24
Default1	Homepage Category Renderer		Update Set: Default	2013-09-24
Table B	Application Menu		Update Set: Default	2013-09-24
sc_catalog_homepage_category_list1	Macro		Update Set: Default	2013-09-24
Hardware	Category		Update Set: Default	2013-09-24

List of changes that are ignored

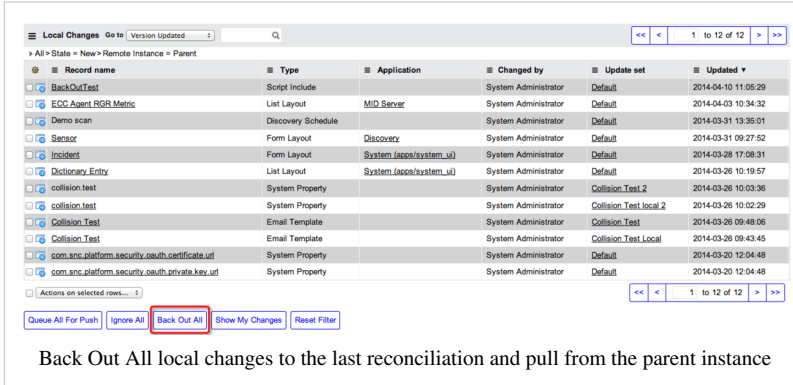
## Backing Out Local Changes

To back out all local changes and restore the last version reconciled with the parent instance (available starting with the Eureka release):

1. Define a parent instance.
2. Pull changes from the parent instance.

3. Navigate to **Team Development > Team Dashboard**.

4. Filter the **Local Changes** list to show only the changes that you want to back out.
5. Do one of the following:
  - Click **Back Out All**.
  - Right-click the local change you want to back out, and then click **Back Out**.



The screenshot shows the 'Local Changes' interface with a table of changes. The table has columns: Record name, Type, Application, Changed by, Update set, and Updated. The 'Back Out All' button is highlighted in red in the bottom left corner of the interface.

Back Out All local changes to the last reconciliation and pull from the parent instance

## Pushing Versions

Pushing promotes changes from the development instance to the parent instance. It commits the current version of a customized record on the development instance as the current version on the parent instance. Pushing adds only the current development version to the parent, not all the development versions.

Pushing creates a local update set on the parent that is marked as complete. Pushed changes are also tracked as local changes on the parent. Therefore, you can promote changes through your development and test hierarchy by transferring the update set or by pushing the local changes. Each push is recorded in the Push or Pull table on the development instance.

**Note:** Updates to records from different applications cannot be pushed/pulled in the same push/pull. To resolve the error in the case that updates to other applications are mixed in:



1. De-queue the updates to other applications.
2. Push for one app.
3. Re-queue the updates to one application.
4. Push and then repeat as needed.

To push versions to the parent instance:

1. Navigate to **Team Development > Team Dashboard**.
2. Queue the local changes that are ready to push.
3. Pull versions from the parent instance and resolve any collisions.

You cannot push changes to the parent instance if collisions are detected.

4. In the control panel, click **Push**. The Push Changes page opens.
5. Provide a **Name** for the changes.
6. Review the list of changes to ensure that the correct changes are included.
  - To remove changes that you do not want to push, select the check boxes beside the rows and select **Do Not Push** from the **Actions** choice list.
  - To add changes, click **Cancel** and repeat the procedure from step 2.

**Push Changes** [Push Changes] [Cancel]

Pushing changes to: <https://demo.service-now.com>

Changes ready to push: **33**

Name: Conference room bookings v1

Comments: Push v1 of Conference room bookings

Changes to Push: Go to [Version Application] [1] to 10 of 33

► All > State = Pending Push > Remote Instance = Dev - Parent

Record name	Type	Application	Source
<input type="checkbox"/> Request Number	Dictionary	Conference room bookings	Update Set: Default
<input type="checkbox"/> REQ	Number	Conference room bookings	Update Set: Default
<input type="checkbox"/> Request	Table	Conference room bookings	Update Set: Default
<input type="checkbox"/> Request	Dictionary	Conference room bookings	Update Set: Default
<input type="checkbox"/> u_conference_room_bookings_user.u_request	Access Roles	Conference room bookings	Update Set: Default
<input type="checkbox"/> u_request	Access Control	Conference room bookings	Update Set: Default
<input type="checkbox"/> u_conference_room_bookings_user.u_request	Access Roles	Conference room bookings	Update Set: Default
<input type="checkbox"/> u_request	Access Control	Conference room bookings	Update Set: Default
<input type="checkbox"/> u_conference_room_bookings_user.u_request	Access Roles	Conference room bookings	Update Set: Default
<input type="checkbox"/> u_request	Access Control	Conference room bookings	Update Set: Default

[Actions on selected rows...]

[Push Changes] [Cancel]

Review the changes before a push

7. [Optional] Edit the name. The name identifies the push record on the development instance and the local update set record on the parent instance.

8. [Optional] Enter comments. The comments are added to the push record on the development instance and the local update set record on the parent instance.

9. Click **Push Changes**. The system initiates a pull to ensure that there are no collisions before the push proceeds.

- If collisions are detected, the push is automatically canceled and you must repeat the

procedure from step 3.

- If no collisions are detected, the changes are staged on the parent instance. On the parent, each version is validated and then committed in the correct order to maintain dependencies between records. For example, a new table is committed before a field on that table to ensure the field is properly created.

**Note:** You cannot push if there is a version conflict between instances or the pushing instance has changes in the **Awaiting Code Review** stage.

10. On the completion page, click **Show Results**.

11. Review the push record for any errors or skipped changes.

- Changes with a state of **Pushed** were committed on the parent instance.
- Changes with a state of **Skipped** were not committed on the parent instance and remain queued as local changes on the development instance.

12. For each skipped change, review the log message to determine why the change was skipped. Develop any changes that are necessary to commit the desired version on the parent instance, and then push them. Some examples of why a change may be skipped include:

- A table does not exist on the parent because it was created when you activated a plugin on the development instance. Ensure the plugin is activated on the parent and push the change again.
- An error occurred during the push. Try to push again.
- The current version is invalid. Revert to a previous version and make the change again to ensure the version is valid.
- An error occurred on the parent during the push. The **Log** field on the push record contains the exception message. Review the system logs on the parent instance and troubleshoot any problems with the instance.

**Push or Pull**

Name:

Conference room bookings v1

Type:

Push

Created:

2013-09-24 17:03:48

Created by:

admin

Remote Instance:

Dev - Parent

Latest version date:

2013-09-24 17:03:48

Stage:

Complete

Comments:

Push v1 of Conference room bookings

### Related Links

[Team Dashboard](#)

Push and Pull Versions

Go to

Version Type

Q

1

to 10 of 33

▶▶▶

▶ Push or Pull = 2013-09-25 00:03:48

	Record name	Type	Application	State	Log
<input type="checkbox"/>	u_request	Access Control	<a href="#">Conference room bookings</a>	Pushed	
<input type="checkbox"/>	u_request	Access Control	<a href="#">Conference room bookings</a>	Pushed	
<input type="checkbox"/>	u_room	Access Control	<a href="#">Conference room bookings</a>	Pushed	
<input type="checkbox"/>	u_request	Access Control	<a href="#">Conference room bookings</a>	Pushed	

Push record

## Approving or Rejecting Pushes

Code reviewers must approve or reject a push from the team development application. Although reviewers can see the individual versions within a push, they must approve or reject the push as a whole (available starting with the Eureka release).

1. Log in to the parent instance that requires code review.
2. Navigate to **Team Development > Code Review Requests**.

3. Select a change in the **Awaiting Code Review** stage.
4. Review the changes in the **Push or Pull Versions** related list.
5. Click **Approve or Reject**.
6. [Optional] Enter review comments in **Comments**. These comments are visible to anyone who can see the Pushes and Pulls history.
7. Click either **Approve** or **Reject**, as appropriate.

Approve or Reject

Enter your comments for approval or rejection. This comment will be useful for developer who submitted this code change.

Comments:

Enter your approval or rejection comments.

Approve

Reject

Cancel



**Note:** The **URL** and **Remote Instance** fields list the address and name of the instance where the change originated.

## Backing Out a Push

To back out a push (available starting with the Fuji release):

1. Navigate to **Team Development > Pushes and Pulls**.
2. Select the push to back out.
3. Click **Back Out**.
4. Click **OK** when the confirmation message appears.

## Comparing Pushed Versions to Local Versions

Code reviewers can compare the pushed versions to the local versions to see the potential effect of incoming changes.

1. Log in to the instance requiring code review.
2. Navigate to **Team Development > Code Review Requests**.
3. Select a change in the **Awaiting Code Review** stage.
4. Review the changes in the **Push or Pull Versions** related list.
5. Right-click a row in the list and click **Compare to Current**. A comparison of the differences between the pushed and local versions appears.

## Checking the Review Status of Pushed Changes

If the parent instance requires pushed changes to undergo code review, changes are placed in the **Awaiting Code Review** stage. If you configure the parent instance to send notifications, it sends the submitting developer a notification when the pushed changes are approved or rejected. Developers can also manually check the status of their pushed changes from the **Pushes and Pulls** module on the submitting instance.

1. Log in to the instance that submitted code for review.
2. Navigate to **Team Development > Pushes and Pulls**.
3. Filter for the push you want to review.
  - Pushes in the **Complete** stage have been approved and applied to the parent instance.
  - Pushes in the **Collided** stage have been rejected because of a collision.
  - Pushes in the **Awaiting Code Review** stage are awaiting review (available starting with the Eureka release).
  - Pushes in the **Code Changes Rejected** stage have been rejected by a reviewer (available starting with the Eureka release).
  - Pushes in the **Code Review Request Cancelled** stage have been cancelled by the submitting developer (available starting with the Eureka release).
4. Click the **Reviews** related list to see the following information.
  - Who submitted a review decision.
  - What the decision was: either approved or rejected
  - What comments if any the reviewer provided.



## Canceling a Code Review Request

Developers can cancel any push they submitted that is in the **Awaiting Code Review** stage. Canceling a request sets the push to the **Code Review Request Cancelled** stage on the submitting instance. The submitting instance retains a version history of the push but the parent instance does not. Developers can cancel code review requests starting with the Eureka release.

1. Log in to the instance that pushed the changes.
2. Navigate to **Team Development > Pushes and Pulls**.
3. Filter for the push you want to cancel.
 

**Note:** You cannot cancel a push that has been approved or rejected.
4. Select the Push or Pull record.
5. Click **Cancel Code Review**.

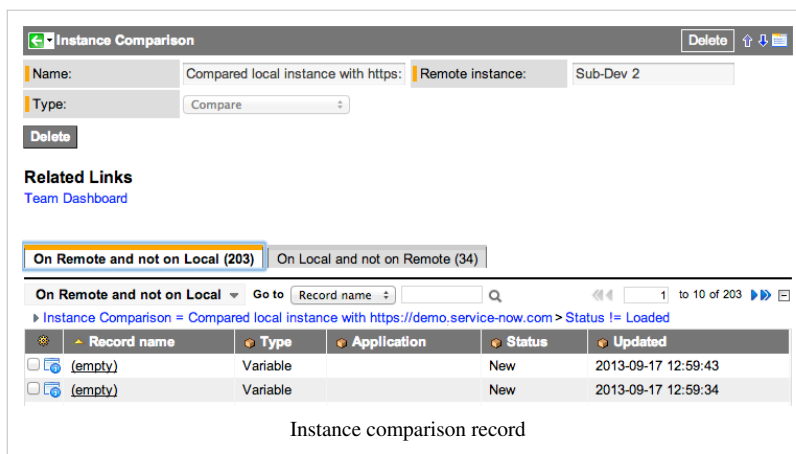
## Comparing to Peer Instances

You can compare the local instance to any other remote instance and commit any current versions from the remote instance on your development instance. Comparing allows you to share code between instances without pushing to a common parent.

Comparing instances does not automatically commit any versions on the local instance. It initiates a full comparison of all changes on the remote instance and all changes on the local instance, and then reports which customized records have different current versions. You can selectively commit a version from the remote instance or compare it with the version on your local instance. You can delete the instance comparison record when you finish evaluating the differences.

To compare the local instance to a peer instance:

1. Ensure the peer instance is defined as a remote instance.
2. Navigate to **Team Development > Team Dashboard**.
3. In the control panel, click **Compare to**.
4. Select the peer instance you want to compare to the local instance and click **Compare**.
5. On the completion page, click **Show Results**. The instance comparison record opens.



6. Review the **On Remote and not Local** related list, which shows the customized records where the current version on the peer instance is not on the local instance. For each customized record, you can:

- Compare the current remote version to the current local version by right-clicking a row and selecting **Compare to Current**.
- Load the current remote version

as the current local version by right-clicking a row and selecting **Load This Change**.

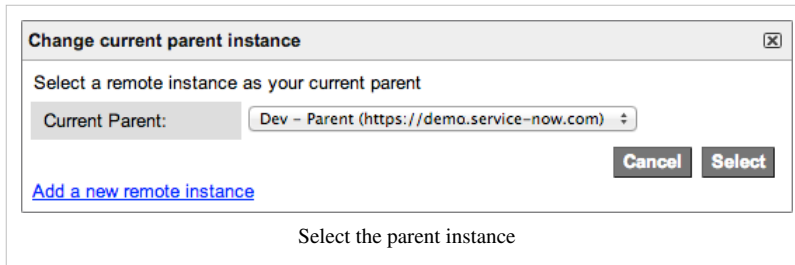
## Changing the Parent Instance

If it becomes necessary to modify the instance hierarchy, you can change the parent for a development instance. Changing the parent initiates a complete comparison between the development instance and the new parent instance. To optimize comparison speed and reduce the number of collisions and local changes that need review afterwards, ensure that the new parent instance was cloned recently from an appropriate instance (for example, the production instance). Before you change the parent instance, ensure that the change does not conflict with your change management process or other development efforts.

To change the parent for a development instance:

1. On the development instance, navigate to **Team Development > Team Dashboard**.
2. In the control panel, click **Change**.
3. Select the remote instance you want to use as the parent and click **Select**.

Alternatively, click the link to define a new remote instance. Then, repeat steps 1–3 and select the remote instance you defined.



**Change current parent instance**

Select a remote instance as your current parent

Current Parent:

[Add a new remote instance](#)

Select the parent instance

The system initiates a reconcile, which compares the local instance to the parent, and then generates the list of local changes and calculates the number of changes that are ready to pull from the parent.

- On the completion page, click **Team Dashboard**.
- Pull versions from the parent instance and resolve any collisions.
- Review the local changes list and queue or ignore changes, as appropriate.

## Reconciling

Reconciling first compares the local instance to the parent, and then generates the list of local changes and calculates the number of changes that are ready to pull from the parent. A reconcile occurs automatically whenever you select a parent instance. You may need to manually reconcile after an external disruptive event on the parent instance, such as a clone or failover.

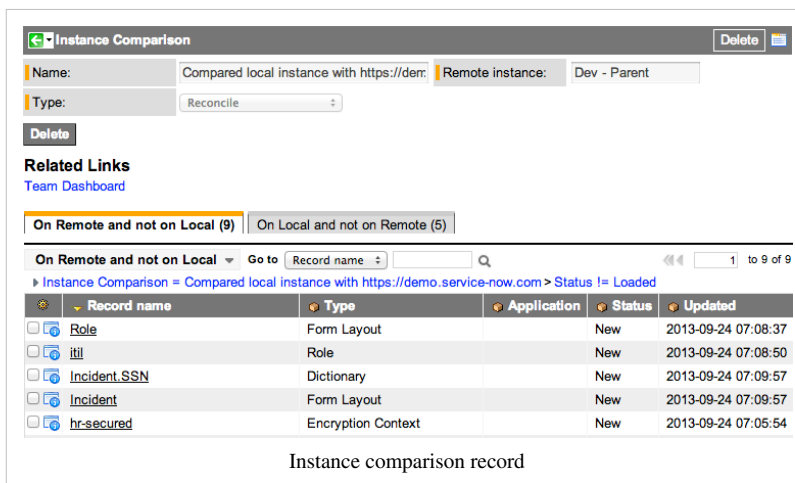


**Note:** This process may take a while to complete depending on the size and age of the instance.

- Navigate to **Team Development > Team Dashboard**.
- In the control panel, click **Reconcile**.
- In the confirmation dialog box, click **OK**.

The list of local changes that have not been queued or ignored is recreated. If you had previously queued or ignored a local change, that designation is maintained.

- [Optional] On the completion page, click **Show Results**. Review the instance comparison record.
  - The **On Remote and not Local** related list shows the versions that are ready to pull from the parent.
  - The **On Local and not on Remote** related list shows the local versions that are ready to queue or ignore.



**Instance Comparison**

Name:  Remote instance:

Type:

**Related Links**

[Team Dashboard](#)

**On Remote and not on Local (9)** **On Local and not on Remote (5)**

Go to   1 to 9 of 9

Instance Comparison = Compared local instance with https://demo.service-now.com > Status != Loaded

Record name	Type	Application	Status	Updated
<input type="checkbox"/> Role	Form Layout		New	2013-09-24 07:08:37
<input type="checkbox"/> itil	Role		New	2013-09-24 07:08:50
<input type="checkbox"/> Incident.SSN	Dictionary		New	2013-09-24 07:09:57
<input type="checkbox"/> Incident	Form Layout		New	2013-09-24 07:09:57
<input type="checkbox"/> hr-secured	Encryption Context		New	2013-09-24 07:05:54

Instance comparison record

- Click **Team Dashboard**.
- Pull versions from the parent instance and resolve any collisions.
- Review the local changes list and queue or ignore changes, as appropriate.

# Versions



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

## Overview

Version records track changes to a customized record over time so that administrators can compare or revert to specific versions later. Administrators can also transfer versions between instances with update sets or team development.

## Version Records

The Update Versions [sys\_update\_version] table contains records that represent the state of a customizable object at a given point in time. A customizable record is any object that is tracked by update sets, such as business rules or script includes. A new version record is created automatically whenever a user changes a customizable record or changes the application file for the customizable record.

A *baseline version* record represents the version of a base system object as it was delivered in the most recent upgrade. Baseline versions are created only for objects that have been modified by a user, and they are updated each time the system is upgraded. In versions prior to the Dublin release, baseline versions are created for every customizable record. Baseline versions allow you to revert customizations to the most recent system version.

Field	Description
Name	A unique identifier for coalescing versions of the same customized record.
Record name	Name of the customized record (starting with the Dublin release).
Source	Indicator of how the version was added on the instance. <ul style="list-style-type: none"> <li>• <b>System Upgrade:</b> from a software upgrade (the baseline version).</li> <li>• <b>Update Set:</b> from an update set that was created or committed on the instance.</li> <li>• <b>Pull History:</b> from a pull in team development (starting with the Dublin release).</li> </ul>
State	Indicator of whether the version is or has ever been loaded on the instance (starting with the Dublin release). <ul style="list-style-type: none"> <li>• <b>Current:</b> the version is currently loaded.</li> <li>• <b>Previous:</b> the version has previously been loaded on the instance. When a current version is replaced by a new version, it becomes a previous version.</li> <li>• <b>History:</b> the version was never loaded on the instance and was only inserted for historical purposes, such as when pulling versions from the parent in team development.</li> </ul>
Application	The application for the customized record, if it is assigned to an application (starting with the Dublin release).
Payload	The data for this version of the customized record.

### Additional fields on the list view

Reverted from	A reference to the older version record, if this version was created by reverting to an older version.
---------------	--

### Fields that can be added by configuring the form

Instance Name	The name of the remote instance where the version was originally created (starting with the Dublin release).
Instance ID	The URL of the remote instance where the version was originally created (starting with the Dublin release).



#### Related lists on the form view

**Version List** All versions of the customized record that are available on the instance (starting with the Dublin release).

## Transferring Versions

Administrators transfer version records between instances by moving customizations with update sets or team development.

- **Update sets:** committing an update set adds versions. For each update in the update set, the version that corresponds to the update is added on the local instance. See [Committing Update Sets](#).
- **Team development:**
  - Pulling retrieves from the parent instance all versions of customized records that have not already be pulled and adds them on the local instance. See [Pulling Versions](#).
  - Pushing adds to the parent instance only the current local version, not all the local versions. See [Pushing Versions](#).
  - Loading changes from peer instances adds selected versions to the local instance. See [Comparing to Peer Instances](#).

## Navigating Version Records

You can view a list of versions for an object by using one of the following methods.

- For forms or lists, right-click the header and select **Configure > Form Layout (Personalize > Form Layout** in versions prior to Fuji) or **Configure > List Layout (Personalize > List Layout** in versions prior to Fuji). Under **Related Links**, click **Show Versions**.
- For tables that use the `update_synch` attribute, add the **Versions** related list to the form. This list is on several forms by default, including, business rules, UI actions, and client scripts.
- For any customizable object, right-click the form header and select **Show Application File**, then scroll down to the **Related Record Versions** related list (starting with the Dublin release).

You can navigate from a version record to:

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

**Update Versions**

Name: content\_block\_iframe\_3088 Record name: Survey

Created: 2013-09-11 07:52:07 Type: IFrames

Source: Push or Pull: 2013-09-11 Action: INSERT\_OR\_UPDATE

State: Current Application:

Payload: XML

```
<?xml version="1.0" encoding="UTF-8"?><record_update table="content_block_iframe">
<content_block_iframe action="INSERT_OR_UPDATE"><active>true</active>
<category>general</category><condition/><conditional>false</conditional></frame/>
```

**Related Links**

[Revert to this version](#)

[Show Related Record](#)

[Show Application File](#)

**Version List**

Created	Name	Source	State
2013-09-11 07:52:07	content_block_iframe_3088aa7c5d730100a92e5d724906ea38	Push or Pull: 2013-09-	Current

Version record

**Related Links**, click **Compare to Current**.

To compare any two versions of an object:

1. View a list of version records for an object.
2. Select the check boxes beside the two versions to compare.

The current version has the most recent creation date. For the current version, the **State** field is highlighted in green (starting with the Dublin release).

SOAP Message Parameters (1) | SOAP Message Tests | Versions (3)

Versions ▾ New Go to Created ▾ Q

Update Versions

Created	Name
2013-09-11 11:08:04	sys_soap_message_function_57c4dbb45db30100a92e5d724f
2013-09-11 11:06:17	sys_soap_message_function_57c4dbb45db30100a92e5d724f
2013-09-11 11:05:15	sys_soap_message_function_57c4dbb45db30100a92e5d724f

Actions on selected rows...

Compare

Delete

Move to Application...

Select the versions to compare

3. In the **Action** choice list, select **Compare**. A comparison of the differences between the two versions appears.

**Compare Versions**

2013-09-11 18:06:17 Revert to this Version 2013-09-11 11:08:04 (Current Version)

sys_updated_on	2013-09-11 18:06:17	2013-09-11 11:08:04
envelope	1: <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">	1: <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:inc="http://www.service-now.com/incident">
	2: <soapenv:Header>	2: <soapenv:Header>
	3: <soapenv:Body>	3: <soapenv:Body>
	4: <inc:getKeys>	4: <inc:getKeys>
	5: <!--Optional-->	5: <!--Optional-->
	6: <active>false</active>	6: <active>\$(active)</active>
	7: </inc:getKeys>	7: </inc:getKeys>
	8: </soapenv:Body>	8: </soapenv:Body>
	9: </soapenv:Envelope>	9: </soapenv:Envelope>

Compare two versions

## Comparing Versions

You can compare versions of any customizable object that a user has modified, such as a form layout or business rule. You also compare the local and pulled version of an object when resolving collisions and when comparing changes in team development. Administrators can suppress versions for specific tables.

To compare a version to the current version of an object:

- From a Versions list, right-click the version and select **Compare to Current**.
- From the Version form, under

4. [Optional] Click **Revert to this Version** to revert to the older version of the object. This option is available only when changes can be reverted (see Reverting Changes).

## Reverting Changes

You can undo changes to a customized record by reverting to an older version.

1. View a list of version records for an object.
2. [Optional] Compare the current version to the older version to ensure you are reverting the desired changes.
3. Right-click the older version and select **Revert to this version**. A confirmation dialog box appears.

If reverting to this version will result in data loss due to a database schema change, a warning message appears in the dialog box (starting with the Dublin release).

4. Click **OK** to confirm the action.

- The current version is marked as a previous version.
- A new version record is added that duplicates the version selected in step 3. This new version is marked as the current version.



**Note:**

- You can revert to the most recent baseline version. You cannot revert to an older baseline version.
- In versions prior to Dublin, you cannot revert database schema changes.

## Suppressing Versions

Administrators can specify tables for which customizations are not tracked in the Versions [sys\_update\_version] table.



**Warning:** If you suppress versions for tables:

- Team development may work incorrectly.
- You cannot compare and revert versions of records on the tables.

To suppress versions for tables:

1. Navigate to **sys\_properties.list**.
2. Create a new property:
  - **Name:** *glide.update.suppress\_update\_version*
  - **Type:** *string*
  - **Value:** a comma-separated list of tables. The default value is **sys\_user,sys\_import\_set\_row**.

## Enhancements

### Dublin

- The **State** field is added to show whether the version is or has ever been loaded on the instance. For the current version of a record, the field is highlighted in green.
- In the **Source** field, the **Pull History** value is added to indicate that the version was added by a pull in team development.
- The **Instance Name** and **Instance ID** fields are added to identify where the version was originally created.
- The **Version List** related list shows all versions of the customized record that are available on the instance.
- To improve upgrade performance, baseline versions are created only for objects that have been modified by a user. They are not created for every system object. During an upgrade, the baseline version is updated for every object that has customer updates, which allows administrators to revert customizations to the most recent system version.
- Changes to versions improve system performance when comparing versions.

## References

- [1] [https://docs.servicenow.com/bundle/jakarta-application-development/page/build/team-development/concept/c\\_Versions.html](https://docs.servicenow.com/bundle/jakarta-application-development/page/build/team-development/concept/c_Versions.html)

# Article Sources and Contributors

**Team Development** *Source:* <http://wiki.servicenow.com/index.php?oldid=250380> *Contributors:* Cheryl.dolan, Debbie.bodinger, Fuji.publishing.user, John.ramos, Michael.randall, Rachel.sienko, Roy.lagemann, Vaughn.romero

**Setting Up Team Development** *Source:* <http://wiki.servicenow.com/index.php?oldid=245599> *Contributors:* Cheryl.dolan, Rachel.sienko, Roy.lagemann, Vaughn.romero

**Using Team Development** *Source:* <http://wiki.servicenow.com/index.php?oldid=248053> *Contributors:* Cheryl.dolan, David.Bailey, Fuji.publishing.user, Joseph.messerschmidt, Rachel.sienko, Roy.lagemann, Vaughn.romero

**Versions** *Source:* <http://wiki.servicenow.com/index.php?oldid=251046> *Contributors:* Cheryl.dolan, David.Bailey, Emily.partridge, Fuji.publishing.user, John.ramos, Rachel.sienko

# Image Sources, Licenses and Contributors

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

**Image:TeamDevProcessFull.png** *Source:* <http://wiki.servicenow.com/index.php?title=File:TeamDevProcessFull.png> *License:* unknown *Contributors:* Debbie.bodinger

**Image:team\_development\_code\_review\_workflow.png** *Source:* [http://wiki.servicenow.com/index.php?title=File:Team\\_development\\_code\\_review\\_workflow.png](http://wiki.servicenow.com/index.php?title=File:Team_development_code_review_workflow.png) *License:* unknown *Contributors:* Maintenance script

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

**Image:TeamDevelopmentApp.png** *Source:* <http://wiki.servicenow.com/index.php?title=File:TeamDevelopmentApp.png> *License:* unknown *Contributors:* Maintenance script, Rachel.sienko

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

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

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

**Image:TeamDashboard.png** *Source:* <http://wiki.servicenow.com/index.php?title=File:TeamDashboard.png> *License:* unknown *Contributors:* Maintenance script, Rachel.sienko

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

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

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

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

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

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

**image:local\_changes\_back\_out\_all.png** *Source:* [http://wiki.servicenow.com/index.php?title=File:Local\\_changes\\_back\\_out\\_all.png](http://wiki.servicenow.com/index.php?title=File:Local_changes_back_out_all.png) *License:* unknown *Contributors:* Maintenance script

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

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

**Image:approve\_or\_reject\_changes.png** *Source:* [http://wiki.servicenow.com/index.php?title=File:Approve\\_or\\_reject\\_changes.png](http://wiki.servicenow.com/index.php?title=File:Approve_or_reject_changes.png) *License:* unknown *Contributors:* Maintenance script

**Image:code\_review\_status.png** *Source:* [http://wiki.servicenow.com/index.php?title=File:Code\\_review\\_status.png](http://wiki.servicenow.com/index.php?title=File:Code_review_status.png) *License:* unknown *Contributors:* Maintenance script

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

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

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

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

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