Snapshot

Plug-in Guide

2.0.0.180467

escenic

A CCI COMPANY

Table of Contents

1 Introduction

The Escenic Content Engine's Snapshot plug-in provides a source code management service for Widget Framework **blueprints**.

A blueprint is a collection of Widget Framework templates that defines the layout of a Widget Framework web site: the templates in a blueprint are used to render the content in a **site publication**. This separation of layout and content means that the same content can easily be given a different look and feel by changing the blueprint used to render a site publication. Conversely, you can easily ensure that a group of related sites have the same look and feel by using the same blueprint to render them. For further information, see the Widget Framework documentation.

The Snapshot source code management service is based on **git**, the popular open source version control system. It uses git's configuration management functionality to provide template developers with simple development, testing and deployment workflows.

The Snapshot plug-in includes an extension to Content Studio, a dashboard which template developers, testers and production editorial staff can use to manage blueprints.

Template developers can (among many other things) use it to:

- **Branch** blueprints: that is, create development copies of blueprints that they can work on without interfering with other developers' work or affecting the production version of the blueprint.
- **Merge** their changes to a blueprint with changes made by other developers in other branches

Testers can use it to:

- Easily switch between different branches of a blueprint for testing purposes

Editorial staff can use it to:

- Easily update the look and feel of publications by switching between blueprints (changing **active blueprint**).

> Although the Snapshot plug-in provides a relatively simple interface to the underlying git functionality, at least some users in an organisation need to have a good understanding of git in order to be able to resolve any issues that may arise. Merge operations can occasionally fail due to conflicts (where two different users have updated the same field of a widget, for example). In such cases the branches will need to be merged using git directly, and the conflicts manually resolved.

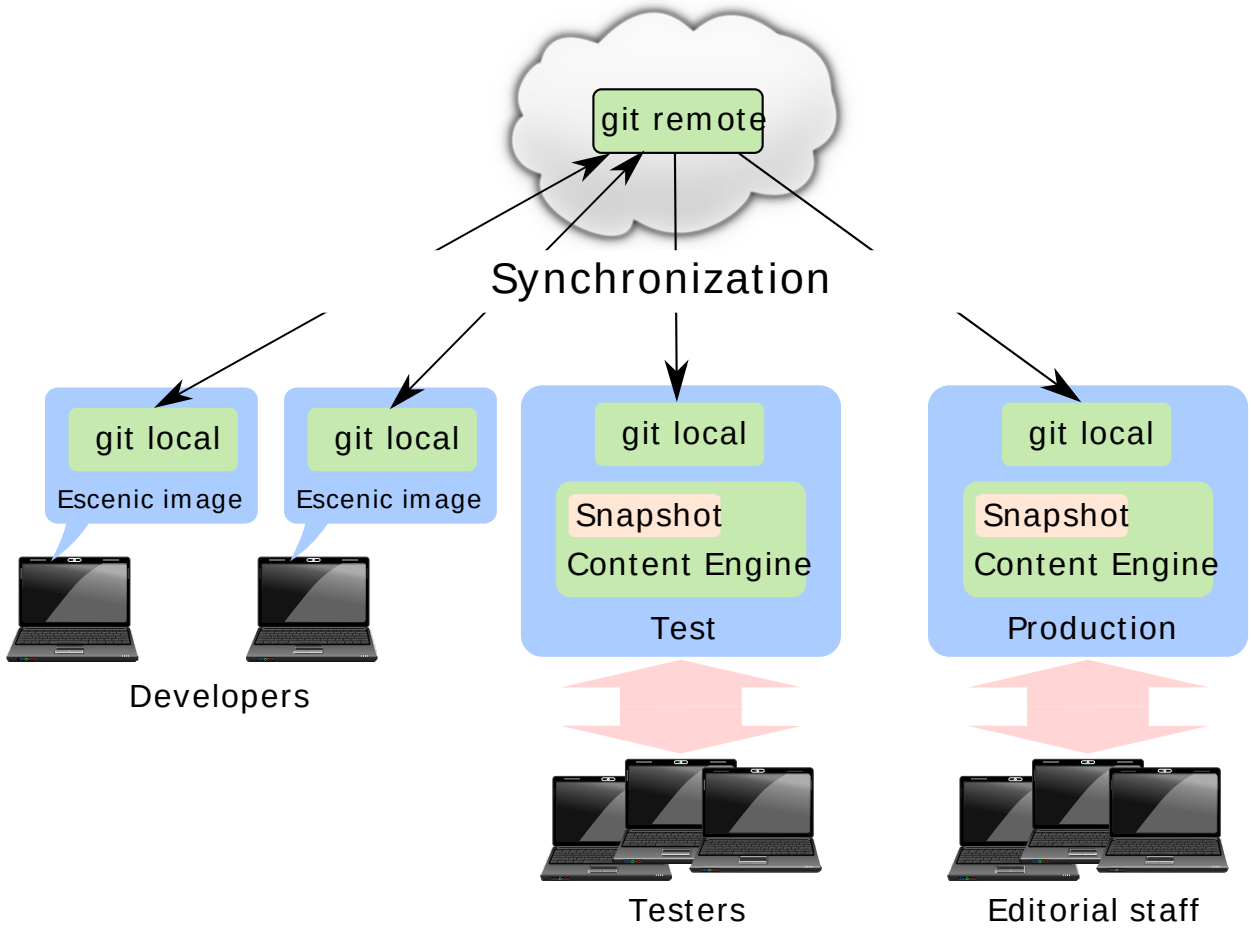
1.1 The Snapshot Plug-in and Git

In order to understand the Snapshot plug-in, you need to understand a little bit about how git works.

Most version control systems prior to git were centralized systems. All users of the system would "check out" a copy of the files they wished to work on from a central repository, and then periodically "check in" the changes they had made. All recording of versions, resolution of conflicts, creation/merging of branches and so on was performed in this central system.

Git is different in that it supports synchronization between repositories. This means you can have many different repositories (or **repos**) containing different versions of the same files, and use git functionality to keep the repos synchronized in whatever way best suits your requirements. It is common practice for every developer working on a project to maintain his own local repo into which she can check in changes, resolve conflicts and create/merge branches. There doesn't **have** to be any central repo, although in most situations it makes sense to have one. When a user wishes to save a set of changes, she does so by committing the changes to her own local repo, and can add a message describing the change. Changes committed to a local repo can be later copied to any other repo (usually some kind of central repo) by synchronizing the repos.

The Snapshot plug-in is designed to support this development model. For each blueprint, it knows about two git repositories: a **local** repository on the Content Engine server and a **remote** repository which holds the central version of the blueprint, accessible to all users. In a typical Escenic-based operation there are three or more parallel installations to consider: **production**, **test** and **development**. In the case of development, each developer usually has a personal server installed as a virtual machine image on his PC. Each Content Engine instance has its own Snapshot plug-in with one or more local repositories containing blueprints. All of the Snapshot plug-ins, however, are configured to communicate with the same remote git server. In this way, all the installations have access to the same, shared remote repositories:



1.2 For the Curious

Blueprints are in fact just Escenic publications that are used to hold widgets and templates rather than sections and content items. Like any other Escenic publications therefore, they are stored in an SQL database – so why do they need to be stored in a git repo as well? And how is it done?

The Snapshot plug-in uses git for storage in order to take advantage of the sophisticated versioning and collaboration functionality that a version control system offers.

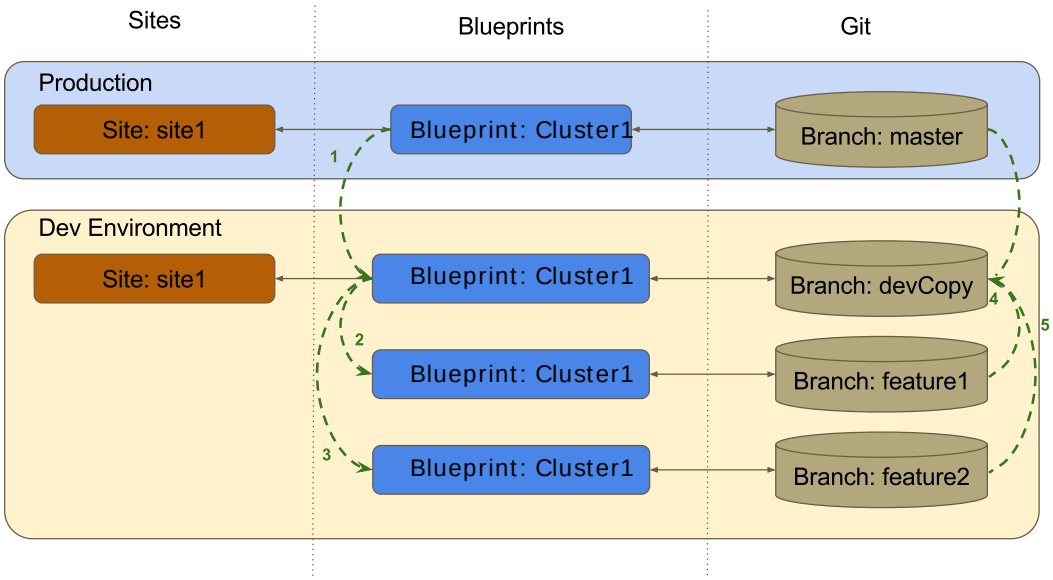
In order to be able to store a blueprint in git, the Snapshot plug-in exports it to a set of Escenic syndication file (that is XML import/export files) behind the scenes, and it is these exported files that are actually saved in the git repo. When the Snapshot user syncs a blueprint with some particular branch, the syndication files in that branch are pulled from the repo and imported into the blueprint in the Content Engine. This means that once you have set up the Snapshot plug-in to take control of your blueprints, the blueprints stored in the Content Engine database are no longer the primary versions, but more like run-time copies.

2 Recommended Workflows

The Snapshot plug-in can support a range of different workflows in order to meet the requirements of different organizations and requirements. This section describes one possible workflow for an organization with three working environments: **Production**, **Development** and **Test**.

The following sections describe the progress of a new development feature through the system. The first diagram shows the development process and the second diagram shows testing and deployment.

2.1 Development



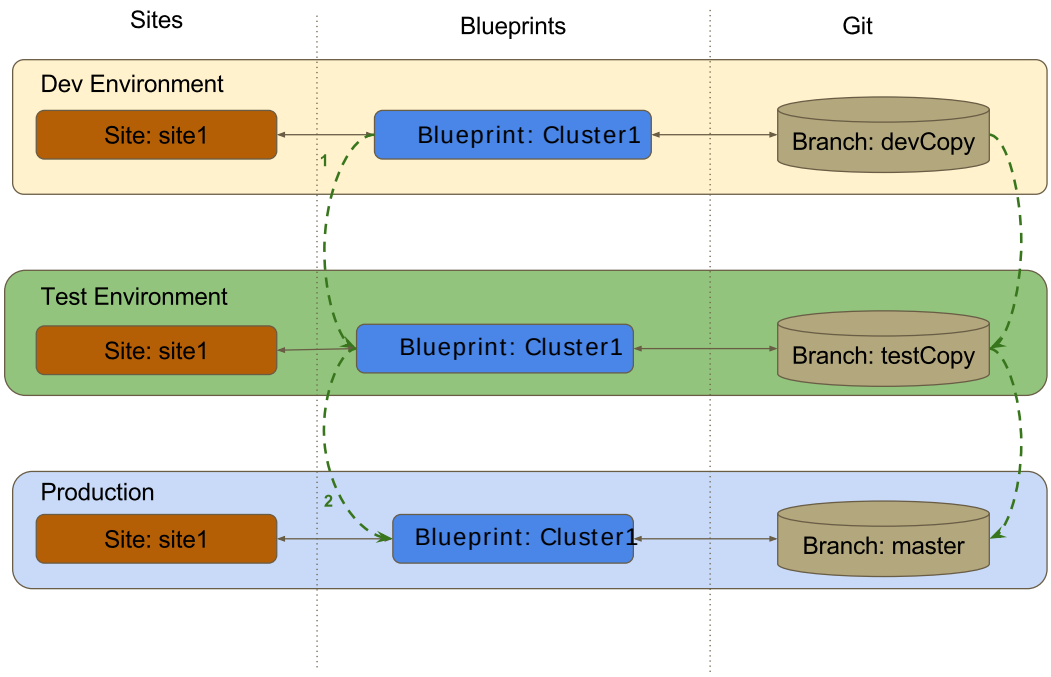
The git **`master`** branch contains a blueprint called **`Cluster1`**. This blueprint is used in production, so it must be kept in a stable condition, and only ever updated with changes that have been fully tested.

1. A development branch called **`devCopy`** is created from the **`master`** branch. This branch will be used to hold sets of changes that can be passed on to the test department for testing.
2. Developer 1 is given the task of adding a new feature to the **`Cluster1`** blueprint. He therefore creates another new branch (**`feature1`**), this time based on the **`devCopy`** branch. Once this branch has been created, the developer can begin working, making changes to the **`Cluster1`** blueprint in the **`feature1`** branch.
3. Developer 2 is given the task of adding another new feature to the **`Cluster1`** blueprint. He therefore creates yet another new branch (**`feature2`**), also based on the **`devCopy`** branch. This developer can then make his changes to the **`Cluster1`** blueprint in the **`feature2`** branch.
4. On finishing his changes, developer 1 merges the **`feature1`** branch back into the **`devCopy`** branch and deletes his working branch (**`feature1`**).
5. On finishing his changes, developer 2 merges the **`feature2`** branch back into the **`devCopy`** branch and deletes his working branch (**`feature2`**). If any of developer 2's changes conflict with the changes made by developer 1, then he will have to resolve the conflicts while making the merge.

The result of this activity is a modified copy of the **Cluster1** blueprint in the **devCopy** branch that contains the changes for two new features. These modifications can therefore be passed on as a unit to the test department for testing.

2.2 Testing and Deployment

The **devCopy** branch now contains a set of features that are ready for testing. The workflow is as follows:



1. A test branch called **testCopy** is created from the **devCopy** branch. This branch contains the two new features that need to be tested.
2. When testing is complete (and if no problems are found), the **testCopy** branch is merged into the **master** branch. This means that the changes to the **Cluster1** blueprint are now available in the production environment and will be visible on the production version of **Site1**.

The advantage of these workflows is that all possible merge conflicts are resolved when feature branches are merged into the **devCopy** branch. Merging the **testCopy** branch back into the production **master** branch can therefore be expected to be a clean, trouble-free operation.

3 Installation

The following preconditions must be met before you can install the Snapshot 2.0.0.180467 plug-in:

- The Content Engine and Escenic assembly tool have been installed as described in the **Escenic Content Engine Installation Guide** and are in working order.
- You have the required plug-in distribution file **`snapshot-2.0.0.180467.zip`**.
- You have access to a git server that can be used for central, shared storage of blueprints. This server can reside anywhere on your network or at some location in the cloud. You can, if you wish, use a public git hosting service such as GitHub or BitBucket. You must ensure that there is a user defined for every host on which you intend to install the Snapshot plug-in. Typically, this would mean one personal user for each developer plus general "test" and "production" users.

3.1 Conventions

The instructions in the following section assume that you have a standard Content Engine installation, as described in the Escenic Content Engine Installation Guide. *escenic-home* is used to refer to the **`/opt/escenic`** folder under which both the Content Engine itself and all plug-ins are installed.

The Content Engine and the software it depends on may be installed on one or several host machines depending on the type of installation required. In order to unambiguously identify the machines on which various installation actions must be carried out, the Escenic Content Engine Installation Guide defines a set of special host names that are used throughout the manual.

Some of these names are also used here:

assembly-host
: The machine used to assemble the various Content Engine components into an enterprise archive or .EAR file.

engine-host
: The machine(s) used to host application servers and Content Engine instances.

editorial-host
: **engine-host**(s) that are used solely for (internal) editorial purposes.

The host names always appear in a bold typeface. If you are installing everything on one host you can, of course, ignore them: you can just do everything on the same machine. If you are creating a larger multi-host installation, then they should help ensure that you do things in the right places.

3.2 Snapshot Installation

Installing Snapshot involves the following steps:

1. Log in as **`escenic`** on your **assembly-host**.
2. Download the Snapshot distribution. If you have a multi-host installation with shared folders as described in the Escenic Content Engine Installation Guide , then it is a good idea to download the distribution to your shared **`/mnt/download`** folder:

```
$ cd /mnt/download
$ wget http://user:password@maven.escenic.com/com/escenic/plugins/snapshot/
snapshot/2.0.0.180467/snapshot-2.0.0.180467.zip
```

Otherwise, download it to some temporary location of your choice.

3. If the folder **`/opt/escenic/engine/plugins`** does not already exist, create it:

   ```
   $ mkdir /opt/escenic/engine/plugins
   ```

4. Unpack the Snapshot distribution file:

   ```
   $ cd /opt/escenic/engine/plugins
   $ unzip /mnt/download/snapshot-2.0.0.180467.zip
   ```

   This will result in the creation of an **`/opt/escenic/engine/plugins/snapshot`** folder.

5. Log in as **`escenic`** on your **assembly-host**.
6. Run the **`ece`** script to re-assemble your Content Engine applications.

   ```
   $ ece assemble
   ```

   This generates an EAR file (**`/var/cache/escenic/engine.ear`**) that you can deploy on all your **engine-host**s.

7. If you have a single-host installation, then skip this step.

   On each **engine-host** on which you wish to run the Snapshot plug-in, copy **`/var/cache/escenic/engine.ear`** from the **assembly-host**. If you have installed an SSH server on the **`assembly-host`** and SSH clients on your**`engine-host`**s, then you can do this as follows:

   ```
   $ scp -r escenic@assembly-host-ip-address:/var/cache/escenic/engine.ear /var/
   cache/escenic/
   ```

   where *assembly-host-ip-address* is the host name or IP address of your **assembly-host**.

8. On each **engine-host** on which you wish to run the Snapshot plug-in, deploy the EAR file and restart the Content Engine by entering:

   ```
   $ ece stop
   $ ece deploy
   $ ece start
   ```

   It only really makes sense to run the Snapshot plug-in on an **editorial-host**.

3.3 Verify the Installation

To verify the status of the Snapshot plug-in, open the Escenic Admin web application (usually located at **`http://`***server*`/escenic-admin`) and click on **View installed plug-ins**. The status of all currently installed plug-ins is shown here, and indicated as follows:

The plug-in is correctly installed.

The plug-in is not correctly installed.

4 Configuration

After installing Snapshot (see chapter 3) you need to configure it to meet your requirements. This involves copying example configuration files from the Snapshot installation into the Content Engine common configuration layer and then modifying the copied files to meet your requirements. In detail, you must:

1. Log in to your Content Engine host as the **escenic** user.
2. Copy all the supplied common configuration layer files as follows:

```
$ cp -r /opt/escenic/engine/plugins/snapshot/misc/siteconfig/com/escenic \
> /etc/escenic/engine/common/com
```

3. Edit the copied files as described in section 4.1 and section 4.2.
4. Assign the Snapshot profiles you have defined to all your existing blueprints and set the initial branch of each blueprint as described in section 4.3.

4.1 SnapshotProfiles.properties

Open **/etc/escenic/engine/common/com/escenic/snapshot/SnapshotProfiles.properties** for editing and set the following properties:

profile.default.sections
This property defines the content of the default Snapshot repo. It is a comma-separated list of section unique names. The listed sections (that is, templates) and all their subsections will be stored in the default Snapshot repo. The default value of **ece_frontpage** selects the entire contents of the blueprint:

```
profile.default.sections=ece_frontpage
```

profile.default.repository
This property specifies the URL of the remote (shared) git repo in which the blueprint is to be stored. If you have set up a repo on BitBucket, for example, you might specify:

```
profile.default.repository=https://my-user@bitbucket.org/my-user/blueprints.git
```

profile.default.snapshot-path
This property specifies location in which the blueprint is to be stored within the git repo. It must be a relative path, but other than that can be anything you like, for example:

```
profile.default.snapshot-path=snapshot/src/main
```

profile.default.username
The user name defined for this profile on the remote git server

```
profile.default.username=user-name
```

profile.default.password
The password defined for this profile on the remote git server

```
profile.default.password=password
```

profile.default.author-name
The author name defined for this profile on the remote git server

```
profile.default.author-name=author-name
```

profile.default.author-email
The email address defined for this profile on the remote git server

```
profile.default.author-email=author-email
```

Together, these properties form a Snapshot **profile** called **default**. One profile corresponds to one remote repo. If you only have one blueprint to manage, then you will only need one repo, and you can just set the three properties listed above. If you have several blueprints that you want to manage, then you can choose either to keep them all in the default repo or to create a separate repo for each blueprint. If you choose to put each blueprint in its own repo, then you will need to define additional profiles by adding similar sets of properties to the file, one set for each profile. You could create a **magazine** profile, for example, by adding the properties **profile.magazine.sections**, **profile.magazine.repository**, **profile.magazine.snapshot-path**, **profile.magazine.username**, **profile.magazine.password**, **profile.magazine.author-name** and **profile.magazine.author-email**.

Once you have created the profiles you want, you must assign them to your blueprints. See section 4.3 for details.

4.2 PublicationConfigHelper.properties

Open **/etc/escenic/engine/common/com/escenic/snapshot/PublicationConfigHelper.properties** for editing and set the following property:

defaultRepoRootLocation
The absolute path of the folder in which the Snapshot plug-in is to create your local repo(s). For example:

```
defaultRepoRootLocation=/var/lib/escenic/snapshot/repository
```

4.3 Configuring the Blueprints

Now that the Snapshot plug-in itself is configured, you need to configure your blueprints so that they will be recognized by the plug-in and handled correctly. To do this you need to add two properties to each blueprint's **feature** resource:

snapshot.profile
The name of the Snapshot profile that is to be used to store this blueprint. If you only have one default profile, then that is what you should specify for all your blueprints, for example:

```
snapshot.profile=default
```

If you have created separate profiles for each blueprint, then you will need to specify different profile names.

snapshot.repo.branch
The name of the branch to which you want this blueprint to be initially committed. You can use whatever branch names you like, but the usual git convention is to call the main branch **master**. Assuming you want to start out with the blueprint in the **master** branch, you should specify:

```
snapshot.repo.branch=master
```

Once you have done this, the Snapshot plug-in is ready for action.

5 Using Snapshot

The Snapshot plug-in adds a tab to the editor area of Content Studio. This tab contains a set of forms that you can use to carry out most of the everyday operations needed to manage your blueprints.

This section contains a brief description of the contents of the Snapshot tab and what the forms can be used for. For detailed instructions on how to perform specific tasks, see chapter 6.

Click on the tab in the editor pane to display the Snapshot forms.

There are three Snapshot forms:

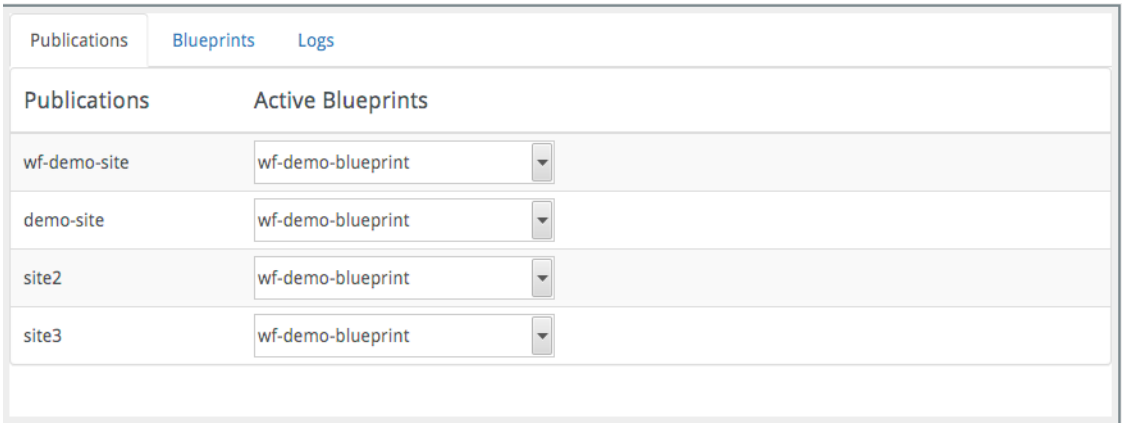
Publications
This form allows you to control which blueprints are used to render your publications.

Blueprints
This form is for managing blueprints, and provides functions for creating and deleting blueprints and branches, committing and merging changes, and so on.

Logs
This forms displays a list of session log messages.

> Whether or not you can use the functions provided by these forms depends on your **role**, and the publication permissions granted to it. For further information about Snapshot permissions, see chapter 7.

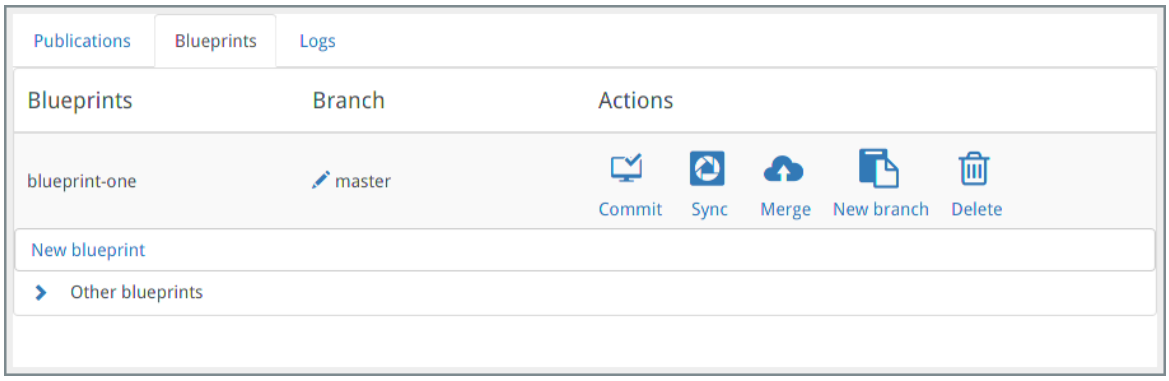
5.1 The Publications Form



This form displays a list of all the site publications for which you have Administrator rights, and their active blueprints (that is, the blueprints that are currently being used to render them). To change a publication's active blueprint, simply select a new blueprint from the drop-down list. Changing the active blueprint in this way will immediately change the appearance of the publication web site.

If you do not have administrator rights for any site publications, then this form will be empty.

5.2 The Blueprints Form



This form displays a list of all the blueprints for which you have administrator rights. For each blueprint, the form displays:

Branch
The branch with which the blueprint is currently synced. To sync with a different branch (that is, to load a different branch from git into the Content Engine), click on the button, click again on the branch name, select the name of the branch you want to load and then click on **Apply**.

Commit
Commits changes you have made to the blueprint to the currently selected branch. A form is displayed listing all differences between your copy of the blueprint, and the copy stored in the local git branch. You can then select the specific changes you want to commit to your local repo.

Sync
Synchronizes the blueprint with the currently selected branch. What this means is that:

- Any uncommitted changes you have made are reverted
- The content of the selected remote branch is copied to the local repo
- The synced blueprint is imported into the Content Engine

You should always close all open editors before syncing a blueprint. If you do not do so then the synced blueprint may fail to be imported into the Content Engine and there will be a mismatch between what you see in Content Studio and what is stored in the local repo.

Merge
This option can be used for two purposes:

- To merge the blueprint in the currently selected branch into another selected branch
- To push the blueprint in your local repo to the same branch in the remote repo

A small form is displayed that you can use to select exactly which branches you want to merge and also whether you want to delete the source blueprint after the merge.

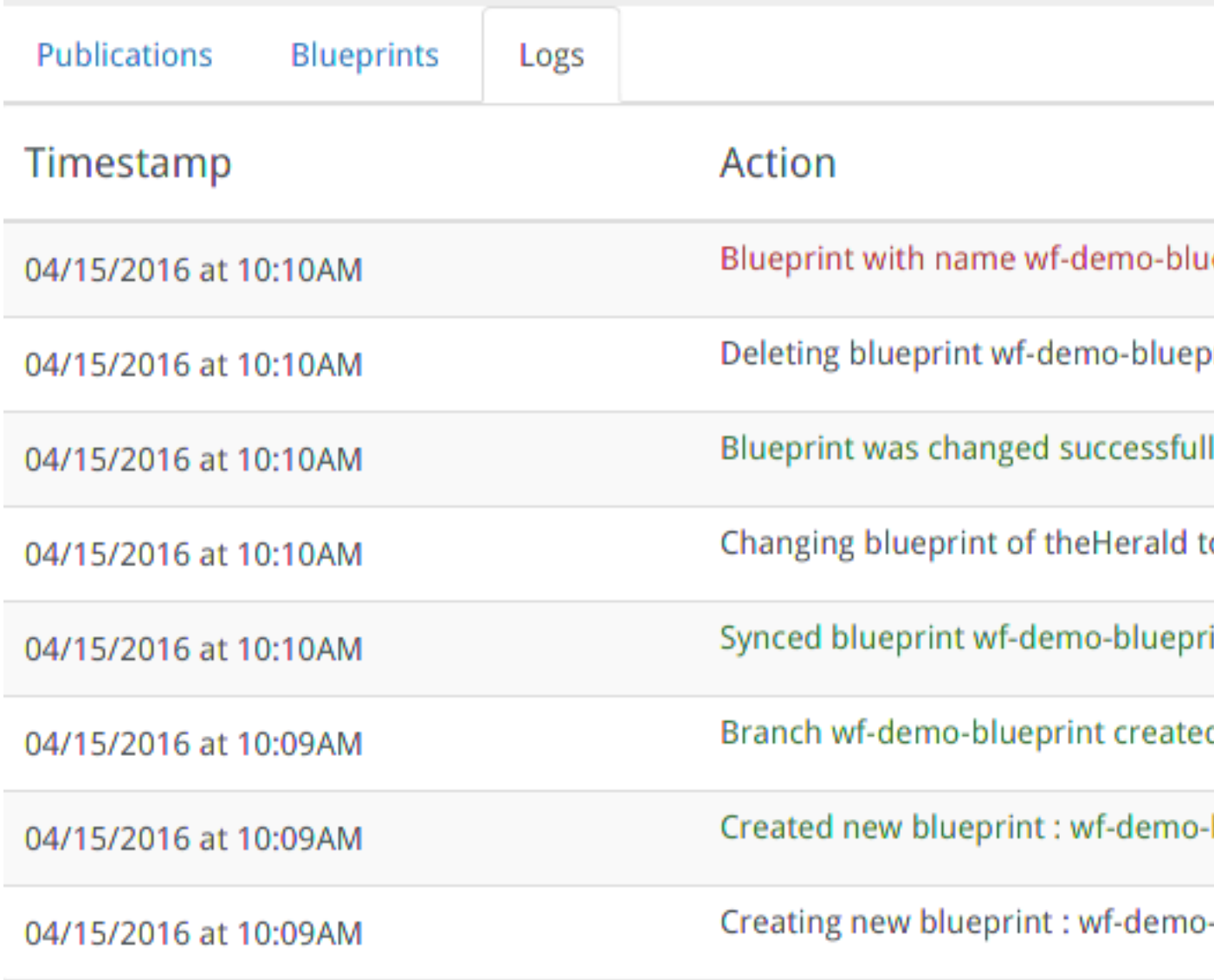
Occasionally, git may be unable to merge the selected branches automatically, due to conflicting changes in the two branches. If this happens, you will need to manually merge the branches using git, and resolve the conflicts. Exactly how you do this is outside the scope of this manual.

New Branch
Creates a new branch of the blueprint, based on a selected source branch. You can choose to either just create the branch in your local repo, or to push it to the remote repo as well.

Delete
Deletes a blueprint. You are asked to confirm it before the blueprint is actually deleted.

There is also a **New blueprint** option displayed at the bottom of the form. This creates a new blueprint by cloning an existing one. You can choose the source blueprint and branch to clone, and specify the name of the new blueprint. You can also choose to create the blueprint in its own branch (which will then have the same name as the blueprint), and whether or not to push the new branch to the remote repo.

5.3 The Logs Form



This form contains a simple list of log messages listing all the Snapshot operations performed in the current session. The list is cleared every time Content Studio is restarted.

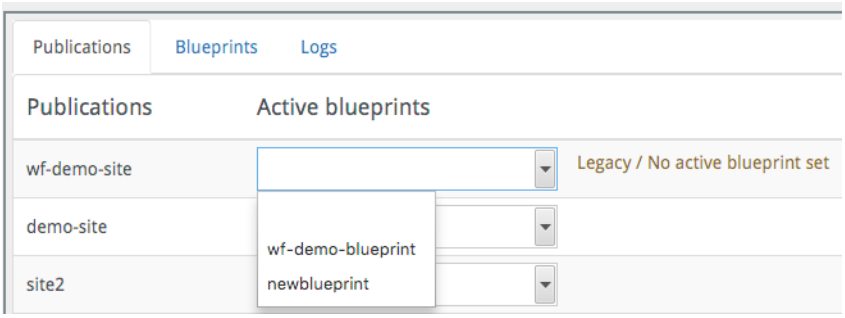
6 How To...

This chapter provides brief descriptions of how to perform common tasks using the Snapshot plug-in.

6.1 Activate a Blueprint

To set or change a publication's active blueprint:

1. Click on the tab in the editor pane to open the Snapshot tab.
2. Click in the **`Active blueprint`** control for the publication you are interested in, and select the required blueprint from the drop-down list:



The Snapshot plug-in immediately applies the selected blueprint to the publication, and the publication's layout is updated accordingly. The plug-in applies the blueprint by updating the **`blueprint.active`** property in the publication's **`feature`** resource.

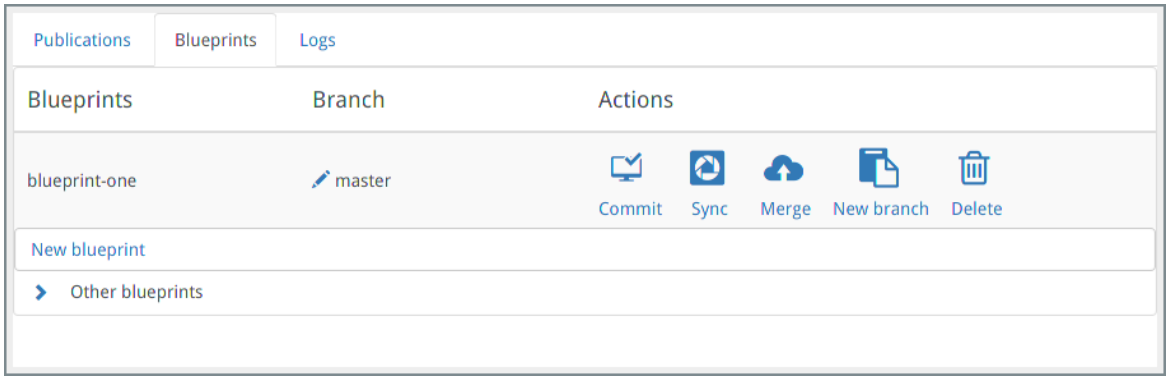
> You can only set the active blueprint of publications for which you have **Administrator** access rights. If you do not have Administrator access rights to any site publications, then the **Publications** list in the above form will be empty.

6.2 Create a New Blueprint

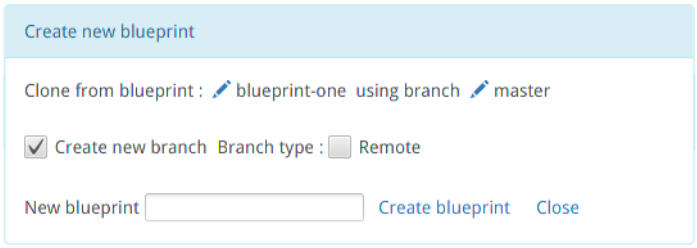
To create a new blueprint:

1. Click on the tab in the editor pane to open the Snapshot tab.

2. Click on **Blueprints**:



3. At the bottom of the form click **New blueprint**:



4. Select:
   - Which blueprint you want to base the new blueprint on (**Clone from blueprint**) by clicking on the icon and selecting the required source blueprint.
   - Which branch of the selected source blueprint you want to base the new blueprint on (**using branch**). You select this the same way.
5. If you want to create the new blueprint in its own branch, then check **Create new branch**. The branch will be given the same name as the new blueprint. If you don't check this option then the new blueprint will be added to the current branch.
6. If you want your new blueprint (and possibly branch) to be pushed to the remote repo where they will be available to everybody, then check **Remote**. Otherwise, the changes will only be made in your local repo.
7. Enter a name for the blueprint in **New blueprint**.
8. Click **Create blueprint**.

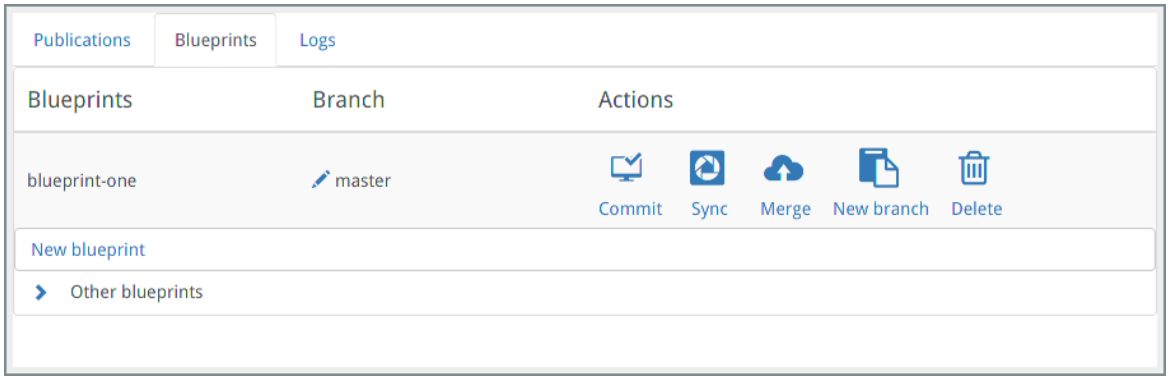
6.3 Change Active Branch

You may have several variants of a blueprint stored in git, in different branches. The Content Engine, however, can only ever contain one version of a blueprint. This is the blueprint's **active** branch: the version you will see if you open the blueprint for editing in Content Studio, or view a publication that uses the blueprint.

To change the active branch of a publication:

1. Make sure all your blueprint editor tabs are closed.
2. Click on the tab in the editor pane to open the Snapshot tab.

3. Click on **Blueprints**:

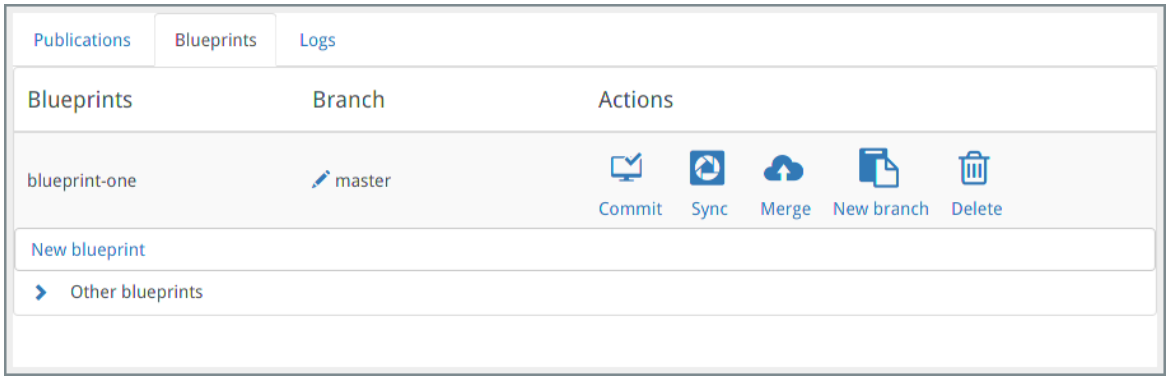


4. If you have made any changes to the current branch that you want to keep, then commit them now (see section 6.4), otherwise you will lose them.
5. Click on the publication's button in the **Branch** column.
6. Select the branch you want to load.
7. Click on **Apply Branch**.

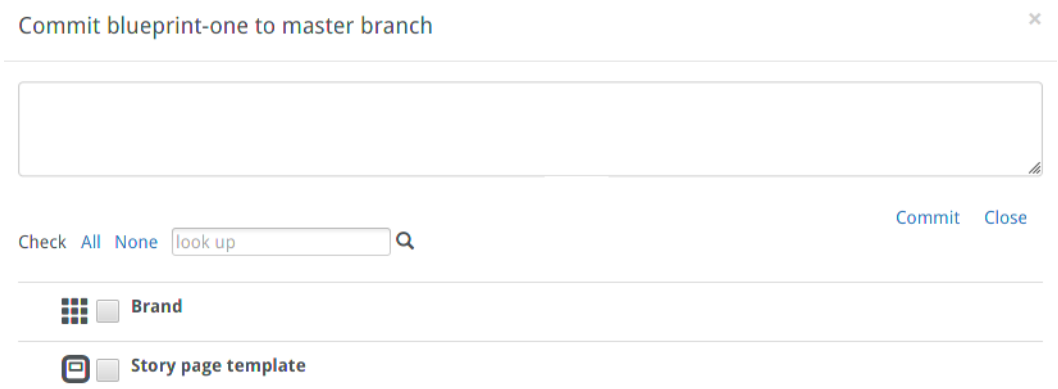
6.4 Commit Changes to a Blueprint

When you save changes to a blueprint in Content Studio, you save the changes in the Content Engine database and nowhere else. In order to save changes in your git repo, you must **commit** them using the Snapshot **Blueprints** form as follows:

1. Click on the tab in the editor pane to display the Snapshot forms.
2. Click on **Blueprints**:



3. Click on the publication's **Commit** button. The following **Commit** form is then displayed:



   This form contains a list of all the uncommitted changes you have made to the branch. "Change" here means changed widget or section. If you have changed several fields in a widget, for example, you will not see one entry for each field, just a single entry for the whole widget.

   Note that if you have previously created new widgets or templates in a different branch and then changed branch, then those new objects will not have been deleted, and will therefore appear in the list of uncommitted changes along with any changes that you actually made in the current branch.

4. Check the changes you actually want to commit.
5. Enter a message describing your changes in the large edit field at the top of form. You must enter a message here, otherwise you will not be allowed to commit anything.
6. Click **Commit**.

Note that committing only commits changes to your local repo. If you are using a personal development image of Content Engine, then the changes will not be visible to any other users. In order for the changes to be visible to others you must push the changes to the global repo using the **Merge** function (see section 6.6).

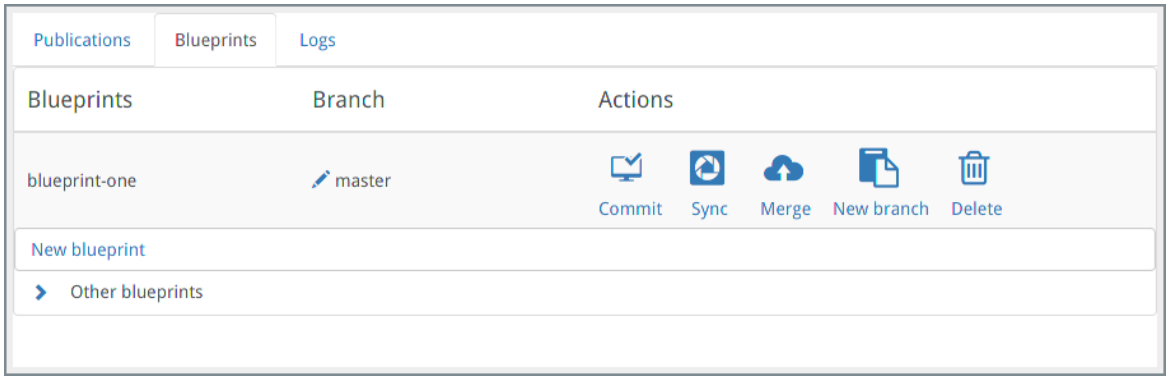
6.5 Sync a Blueprint

The **Sync** function updates your version of the blueprint with any changes made to this branch by other users. It pulls changes from the remote repo (if this branch exists in the remote repo, and then loads the result to the Content Engine.

To sync a blueprint:

1. Make sure all your blueprint editor tabs are closed.
2. Click on the tab in the editor pane to open the Snapshot tab.

3. Click on **Blueprints**:



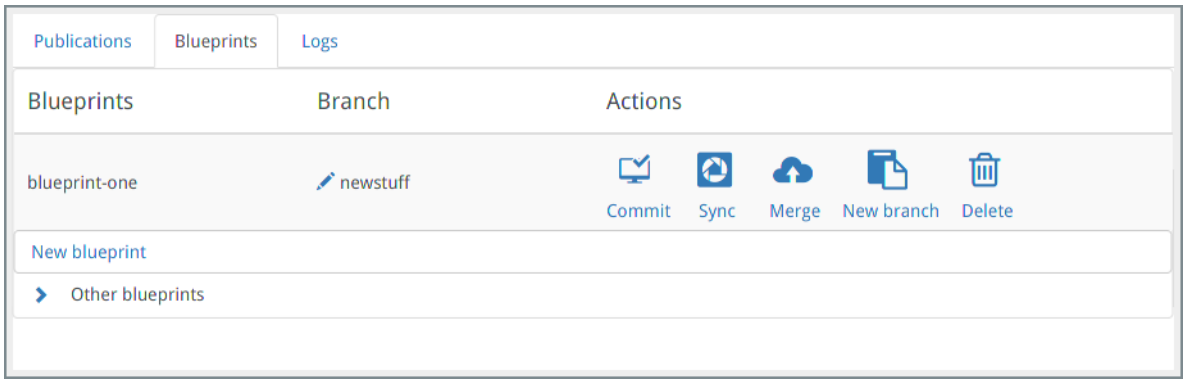
4. If you have made any changes to the blueprint that you want to keep, then commit them now (see section 6.4), otherwise you will lose them. Actually, only changes to existing widgets or templates will be lost. If you have created any new widgets or templates, they will not be deleted by the sync operation.
5. Click on the publication's **Sync** button.
6. If there are any changes to the blueprint that you have not committed, then a warning is displayed. You can either ignore the warning and continue or cancel the operation. If you continue and sync the branch, then the uncommitted changes will be lost.

6.6 Merge a Blueprint

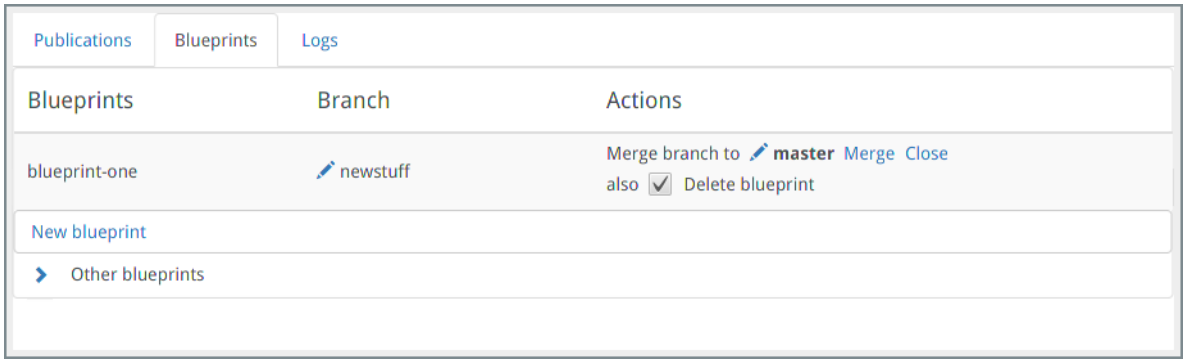
The **Merge** function is used to merge a blueprint from one branch to another (including merging from a branch in the local repo to the same branch in the remote repo).

To merge a blueprint:

1. Click on the tab in the editor pane to open the Snapshot tab.
2. Click **Blueprints**:



3. Click **Merge**. Some new options are then displayed in the **Actions** column of the form:



4. Click on the button and select the target branch into which you want to merge the blueprint. You can merge into the current branch - all you are doing then is merging the changes into the remote repository. (This is called a "push" operation in git.)
5. Check **Delete blueprint** if you have finished with the blueprint you are working on: it will then be deleted once the merge has been carried out (as long as it isn't being used by a site publication).
6. Click **Merge** to proceed with the merge operation, or **Close** to cancel.

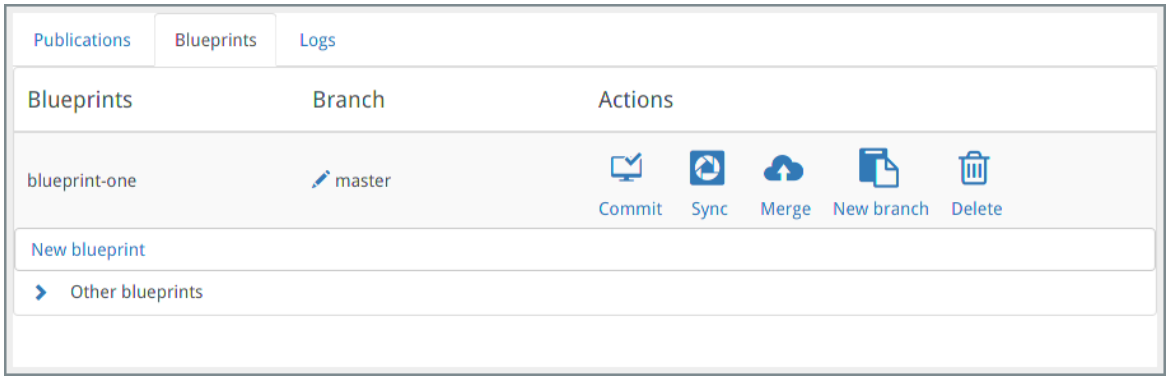
Occasionally, git may be unable to merge the selected branches automatically, due to conflicting changes in the two branches. If this happens, you will need to manually merge the branches using git, and resolve the conflicts. Exactly how you do this is outside the scope of this manual.

6.7 Create a New Branch

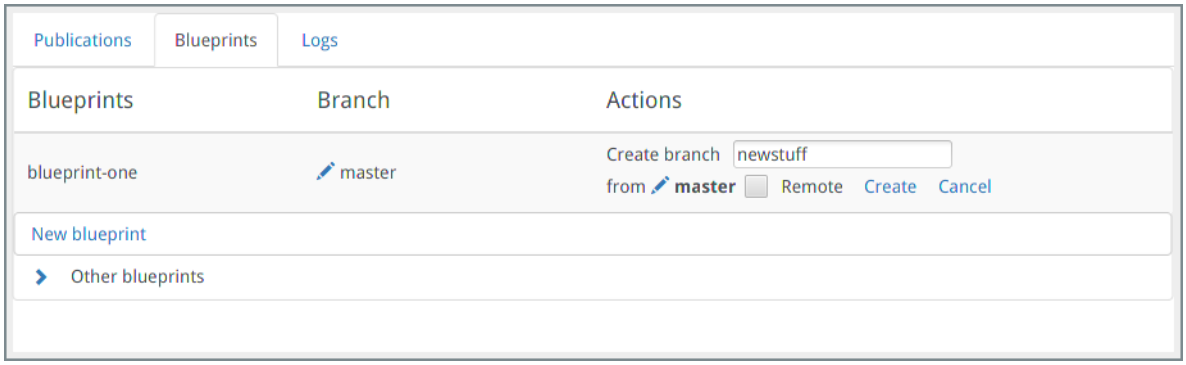
The **Create branch** function lets you create a new branch of a blueprint from any existing branch. You can choose to create just a local branch, or to create a remote copy of it as well.

To create a new branch:

1. Click on the tab in the editor pane to open the Snapshot tab.
2. Click **Blueprints**:



3. Click **New branch**. Some new options are then displayed in the **Actions** column of the form:



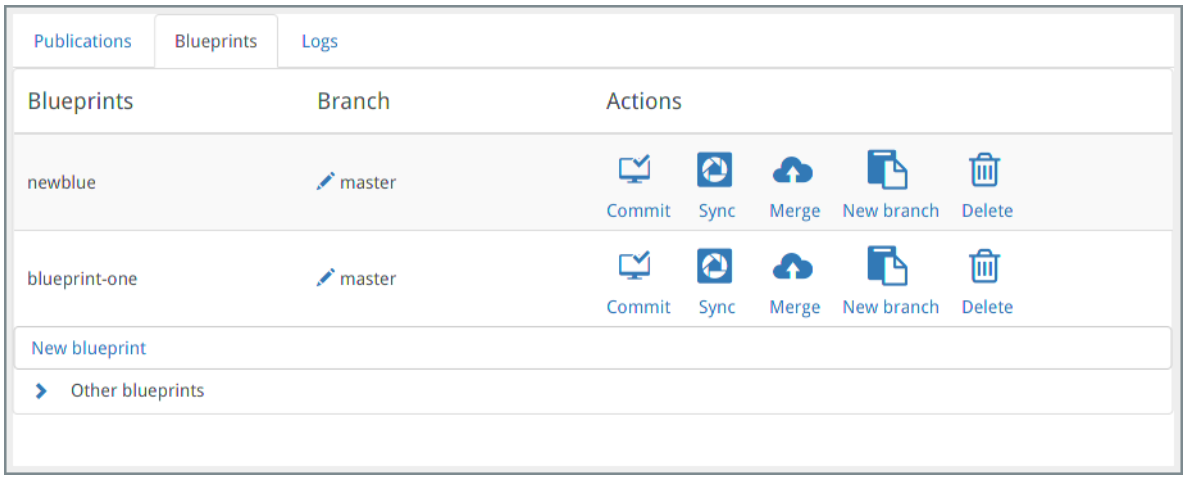
4. Enter the name of the new branch in the **Create branch** field.
5. Click on and select the branch you want to base the new branch on.
6. If you want to create a remote copy of your branch, check **Remote**.
7. Click **Create** to proceed with the create operation, or **Close** to cancel.

6.8 Delete a Blueprint

The **Delete** function deletes a blueprint from the active branch, so long as it is not being used by a site publication. It only deletes the blueprint from the local file system and the Content Engine database. It does not delete anything from either the local or the remote git repo.

To delete a blueprint:

1. Click on the tab in the editor pane to open the Snapshot tab.
2. In the **Publications** form, make sure that the blueprint is not being used by any publications.
3. Click **Blueprints**:



4. Click on the publication's **Delete** button in the **Actions** column.
5. Click on **Yes**.

7 Roles & Permissions

Exactly what you can do with the Snapshot and what is visible in the Snapshot tab depends upon the **roles** you have been assigned. For general information about Content Engine roles and permissions, see Global Roles and Section Roles.

The Snapshot plug-in recognises three different categories of user:

- Site administrator (a user with the **Administrator** role for a site publication).
- Blueprint adminstrator (a user with the **Administrator** role for a blueprint).
- Other users

The Snapshot tab's **Publications** page only contains entries for site publications where you have the **Administrator** role. You can therefore only change a site publication's active blueprint if you are one of the site's administrator.

The Snapshot tab's **Blueprints** page only contains entries for blueprints where you have the **Administrator** role. Therefore you cannot do anything to a blueprint unless you are one of the blueprint's administrators.