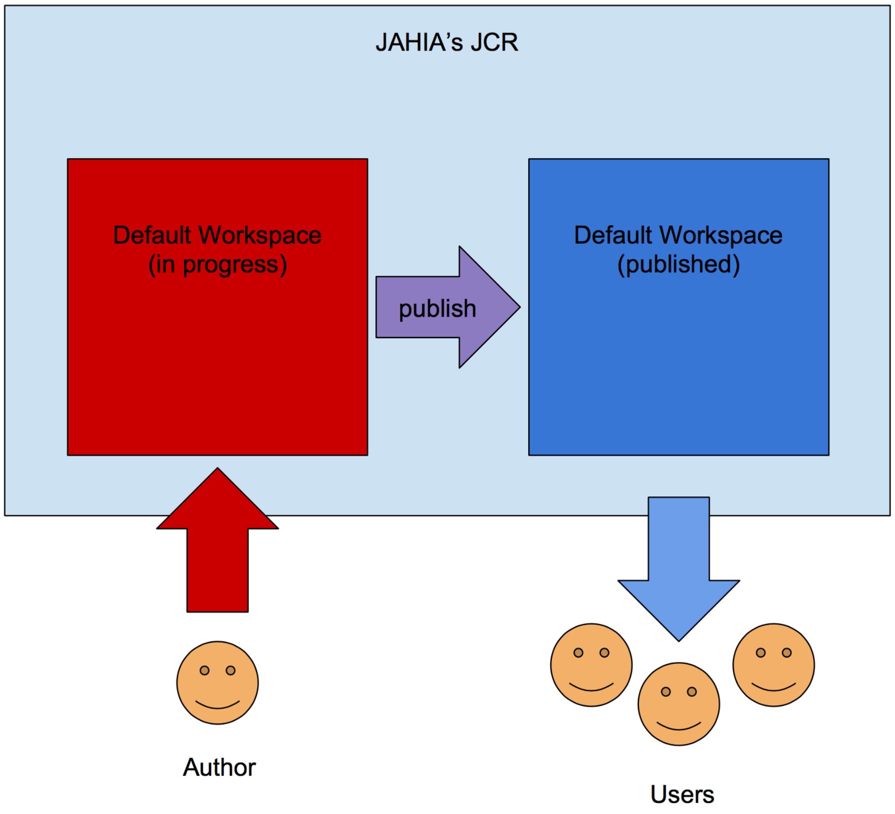
Jahia's JCR

Jahia utilizes a hierarchical database to store its data, the name of the JCR implementation is called Apache Jackrabbit, you may give it a look before continuing reading by following the link below:

http://jackrabbit.apache.org/jcr/index.html

You should already be familiar with the tree structure of a JCR. What you may not be aware is that the repository may have more than one workspace. Each workspace may have completely different contents and/or structure. Jahia stores its contents in two different workspaces, one is called the **default** workspace and the other one is called the **live** workspace.

Those two workspaces work in tandem to store work in progress content (default workspace) and already published content (live workspace): Authors will continue updating the **default** content and when ready, they may publish their work (usually through a workflow) to the **live** workspace.


JAHIA's JCR
Default Workspace (in progress)
publish
Default Workspace (published)
Author
Users

The underlaying storage of the repository is done through a Relational Data Base, the default DB used is MySQL. You should not access the RDB unless something went very wrong and you know what you are doing. Keep the transactions and operations at the JCR level and be sure to know ways of sanitizing it without endangering the integrity of the repository.

Jahias structured model

Jahia is more delicate than some other CMS that use a JCR for storage. The reason being not following David's Model (https://wiki.apache.org/jackrabbit/DavidsModel).

If you follow David's model you learn to be very flexible about your structure. Data drives the structure and not the other way around. Unfortunately Jahia relies on having a proper structure from the start for each component created.

All of these means that when working with Jahia you'll encounter problems when modifying a component that already has content and even worst, it may be already published. If you were to modify the project structure, some inconsistencies may occur in the repository, be aware of them.

JCR Troubleshooting

Deletion of a nodetype

Issue

Deletion of nodetypes could lead to a problem when importing/exporting content.

Even when the nodetype has been removed, previous content may still exist in the source instance. Thus, when exporting such site to another instance without the nodetype, the target instance won't know what to do with the imported content that has such nodetype.

Solution

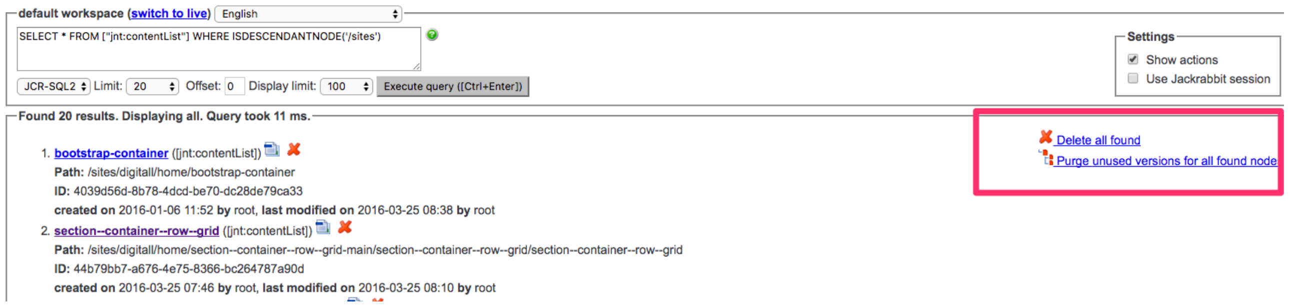
Query for the nodetype definition you removed and remove from the content as well:

For example, imagine you want to remove the nodetype called **jnt:contentList**. First remove the nodetype from the project definition. Afterwards go to the **Jahia's Tools--> JCR Query Tool** (http://localhost:8080/modules/tools/jcrQuery.jsp)

Then you will run the query to find all the content with the **jnt:contentList** nodetype. Like the query below.

```
SELECT * FROM ["jnt:contentList"] WHERE ISDESCENDANTNODE('/sites'
```

Afterwards, delete all nodes found by clicking that option on the top right link:



default workspace (switch to live) English

SELECT * FROM ["jnt:contentList"] WHERE ISDESCENDANTNODE('/sites')

JCR-SQL2 Limit: 20 Offset: 0 Display limit: 100 Execute query ([Ctrl+Enter])

Settings
- [✓] Show actions
- [ ] Use Jackrabbit session

Found 20 results. Displaying all. Query took 11 ms.

Delete all found
Purge unused versions for all found node

1. bootstrap-container ([jnt:contentList])
   Path: /sites/digitall/home/bootstrap-container
   ID: 4039d56d-8b78-4dcd-be70-dc28de79ca33
   created on 2016-01-06 11:52 **by** root, **last modified on** 2016-03-25 08:38 **by** root
2. section--container--row--grid ([jnt:contentList])
   Path: /sites/digitall/home/section--container--row--grid-main/section--container--row--grid/section--container--row--grid
   ID: 44b79bb7-a676-4e75-8366-bc264787a90d
   created on 2016-03-25 07:46 **by** root, **last modified on** 2016-03-25 08:10 **by** root

Nodes being Locked

Issue

When starting workflows (as when publishing), some nodes may be locked. If the workflow fails, the node may stay lock and uneditable:

Solution

Access the JCR Browser (http://localhost:8080/modules/tools/jcrBrowser.jsp).

Then find and remove the following properties from the locked node:

- j:lockTypes
- j:locktoken
- jcr:lockIsDeep
- jcr:lockOwner

The **mandatory** property.

Issue

The use of the **mandatory** property may cause problems. Unless you really need to use it, avoid it. Try to handle missing properties during the rendering process instead of obliging the author to add them, Layer X should already handle this seamlessly, just be aware of it.

First Solution

In cases where a **mandatory** property is added, but eventually won't be needed, make sure to keep the old property (by hiding it) instead of deleting it; since Jahia will expect the **mandatory** property even though it has been removed (this could happen in the **default** or **live** workspace).

Second Solution

If you want to remove the property for good, you'll have to find the mandatory property in the JCR and remove **ALL** instances of it. The problem is that even when no new instances of the component are created, when interacting when already created components, Jahia will expect the mandatory property when other operations are being conducted (like publishing).

Use the query tool once again (http://localhost:8080/modules/tools/jcrQuery.jsp) to find such property and remove it from your content.

Repository desynchronization

Issue

If you are doing modifications on content which is already published, make sure to perform the changes on both repositories: live and default. Otherwise, you will create a desynchronisation that could block any publication of this content.

Solution

If you encounter a synchronization issue between workspaces, it is possible to fix it by exporting the site, reimporting it, and publishing it.