Subversion ALM

Painless and powerful ALM integration of JIRA with Subversion

Contents

[Introduction 4](#_Toc387961501)

[Major Features 4](#_Toc387961502)

[Installation & Configuration 5](#_Toc387961503)

[Install the add-on jar file 5](#_Toc387961504)

[Compatibility 5](#_Toc387961505)

[Accessing to the configuration page 5](#_Toc387961506)

[Registering Subversion repositories 7](#_Toc387961507)

[The Indexing Process 9](#_Toc387961508)

[Configuration parameters 10](#_Toc387961509)

[Re-initializing the Subversion ALM configuration 10](#_Toc387961510)

[Exporting and importing the registered repositories 10](#_Toc387961511)

[The Subversion timeouts 11](#_Toc387961512)

[The pool size 11](#_Toc387961513)

[The Database Web Console 11](#_Toc387961514)

[Compacting the database file size 13](#_Toc387961515)

[Displaying Subversion commits on JIRA 14](#_Toc387961516)

[Displaying Subversion commits on Agile (Greenhopper) 16](#_Toc387961517)

[The Web Client for Subversion 17](#_Toc387961518)

[JIRA integration 17](#_Toc387961519)

[Subversion Web Client in *Standalone* mode 18](#_Toc387961520)

[Major features 19](#_Toc387961521)

[Repository browse 19](#_Toc387961522)

[Revision List 19](#_Toc387961523)

[Revision changes 20](#_Toc387961524)

[Compare files 20](#_Toc387961525)

[Commit graphs 21](#_Toc387961526)

[Statistics 21](#_Toc387961527)

[Reporting 22](#_Toc387961528)

[JQL functions 22](#_Toc387961529)

[Attribute JQL functions 23](#_Toc387961530)

[Aggregation JQL functions 27](#_Toc387961531)

[Commits Calendar Report 29](#_Toc387961532)

[Extending the functionality with 3rd party plugins 32](#_Toc387961533)

[How to use the JDBC connections from a 3rd party plug-in 32](#_Toc387961534)

[Example 34](#_Toc387961535)

[Security 35](#_Toc387961536)

[Audit 36](#_Toc387961537)

[Subscribing to Commit Events 37](#_Toc387961538)

[Office and PDF documents 40](#_Toc387961539)

# Introduction

It provides a painless and powerful integration of JIRA with Subversion. All the web applications required in order to enhance Subversion traceability are bundled with the plugin. They have been fully customized to support the JIRA plugin environment and fully integrated out of the box. The bundled applications are:

* Atlassian JIRA Subversion plugin
* Polarion Web Client for Subversion

With Subversion ALM, JIRA users are able to trace relationships among issues and commits from a single point, explore Subversion repositories and create powerful reports on JIRA by using JQL fully integrated with Subversion.

# Major Features

* Painless installation and configuration
* Multi domain repository (Subversion repositories can be located anywhere)
* Multi connection protocol: file, http, https, svn and svn+ssh
* Displays commits related to issues and projects on JIRA
* Bundled a customized Polarion’s Web Client for Subversion fully integrated out of the box
* Full Commit Graphs providing the backward and forward history of any artifact with support for JIRA issues
* Real time Subversion statistics for users and artifacts (commit calendar views)
* JQL Functions supporting Subversion attributes
* Internal Office and PDF HTML 5viewer

# Installation & Configuration

## Install the add-on jar file

It supports the Atlassian V2 plugin model, hence no JIRA re-start is required. The plug-in is available to download from the Marketplace. It can be installed by following any of the usual methods:

* From within JIRA (Administration > add-ons > Find new add-ons)
* By downloading the binaries (.jar file) from the Marketplace and uploading it into JIRA: Administration > Manage > Installed > Upload a new add-on

## Compatibility

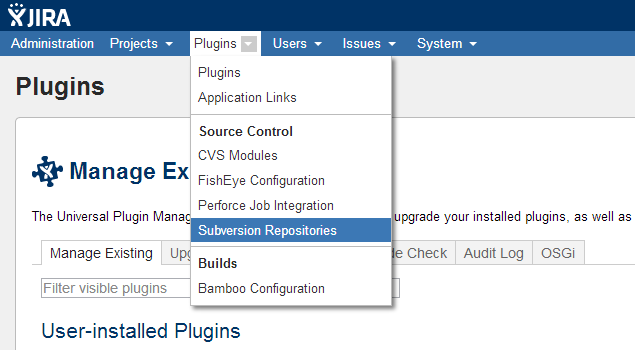
It is fully compatible with the Atlassian’s JIRA Subversion plug-in as well as Fisheye. As Subversion ALM includes a improved version of the Atlassian’s JIRA Subversion plug-in it has not sense keep them installed on the same JIRA instance unless during the evaluation period. Regarding Fisheye, you would get a lot of features overlapping, but while FishEye puts the focus on reposting, Subversion ALM put the focus on traceability.

## Accessing to the configuration page

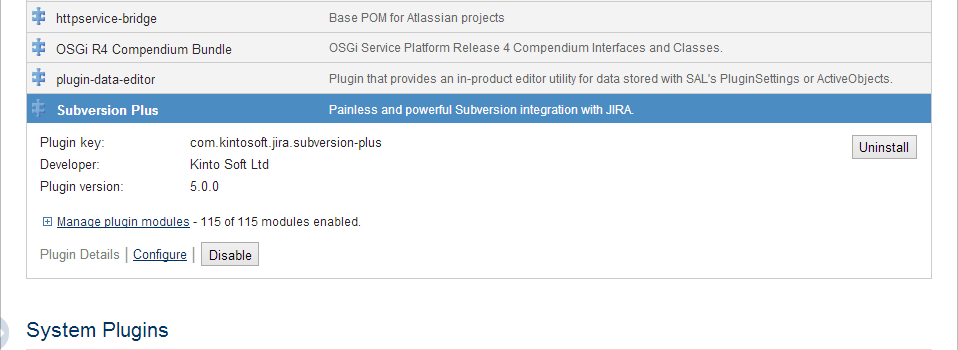
The Subversion ALM configuration page on JIRA requires *Administration* permission.

The configuration page can be accessed from two locations:

1. The Administration > Add-ons > Source Control > *Subversion Repositories* menu:

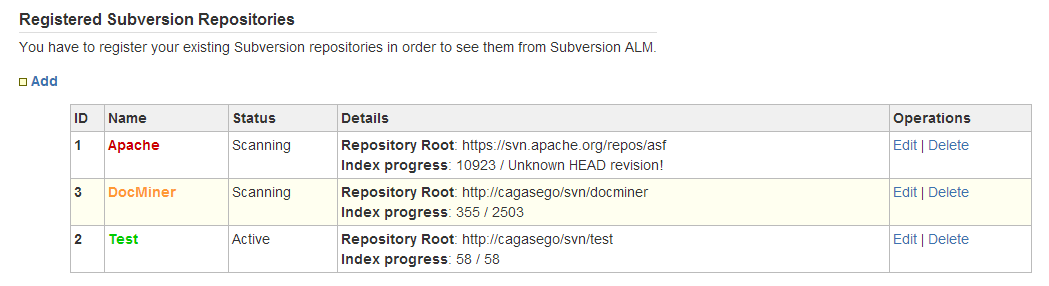
****

From the Administration > add-ons > manage > Installed > *Subversion ALM* > *Configure* link:



## Registering Subversion repositories

Subversion ALM supports any Subversion 5.0+ repository version and all the connection protocols (file, http, https, svn, svn+ssh). The repositories have to be registered in the *Registered Subversion Repositories* panel at the bottom of the configuration page on JIRA. The table lists the already registered repositories:



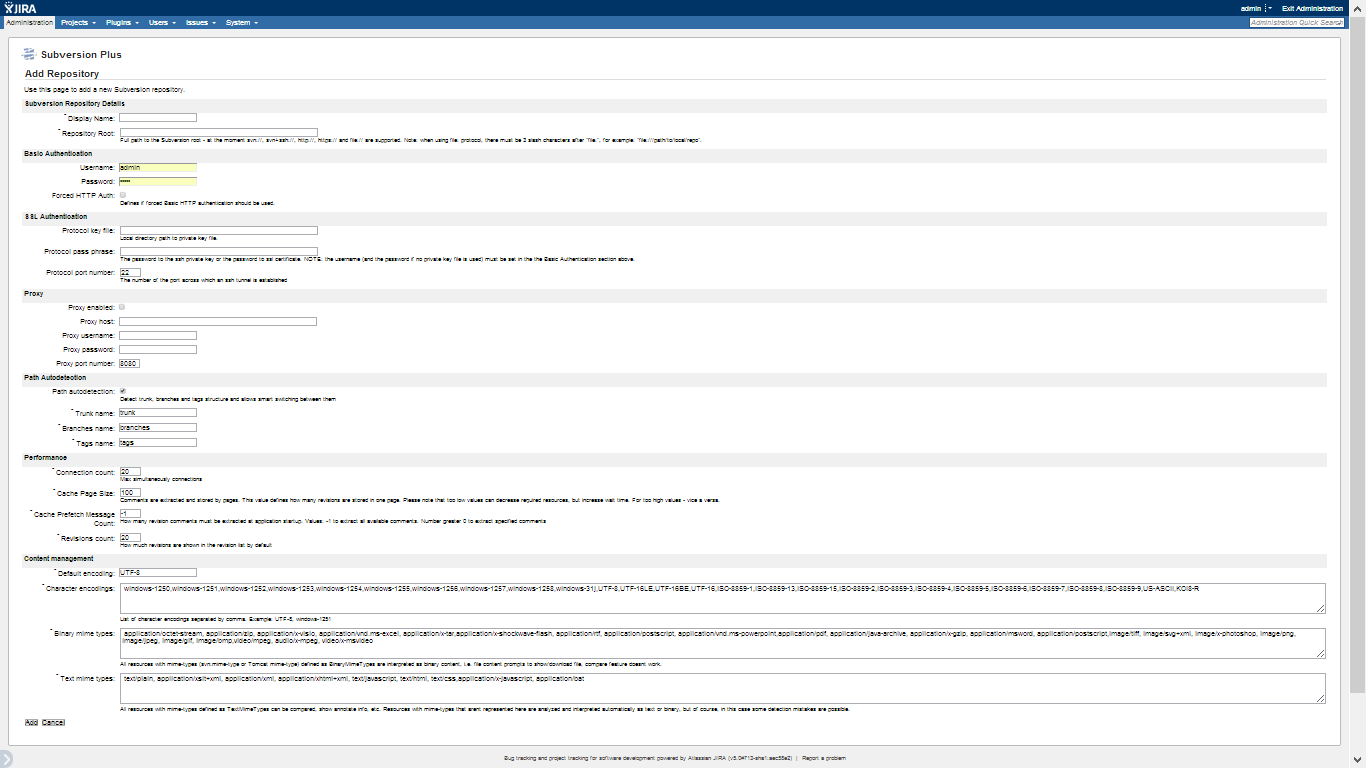
Each repository has:

* **ID**: an unique integer > 0 assigned by the database to each repository automatically.
* **Name**: it will be displayed at the GUI front end to help users to easily identify the repository (so, please, use user friendly names). The color name varies accordingly the indexing progress:
  + ***Red:*** *Error*.
  + ***Orange:*** *In progress*, the *latest indexed* revision is behind the repository HEAD revision.
  + ***Green*:** *Up to date*.
* **Status**:
  + ***Scanning*:** There is a thread fetching the log history for the repository and the repository data are being cached on the database.
  + ***Active*:** No error. The repository is accessible but there is not a thread scanning it because it is up to date or queued (see *MAX\_INDEXING\_THREADS* parameter).
  + ***Inactive*:** Some error has occurred and the repository is not accessible.
* ***Details***:
* **Repository Root**
* **Index progress**: latest indexed revision vs total revisions in the repository (HEAD).

The *Operation* columns show the available actions for each repository. Usually they are:

* *Edit*: allows changing most of the repository configuration data.
* *Delete:* It un-registers the repository and deletes all its data from the H2 database.

To register a new Subversion repository click on the *Add* link. It will open a new page asking for the repository configuration data:

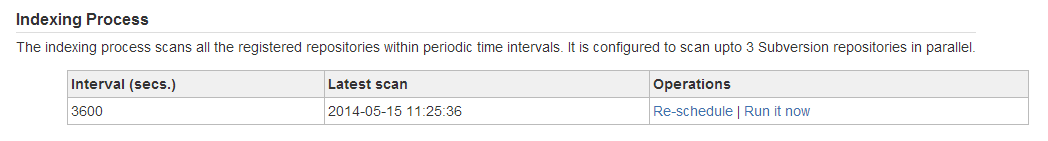


Most of the configuration is automatically populate with default values. Hence, the amount of the required data to be filled is really small: the repository *name*, the *url*, username *credentials* and the connection *protocol* only in most cases.

## The Indexing Process

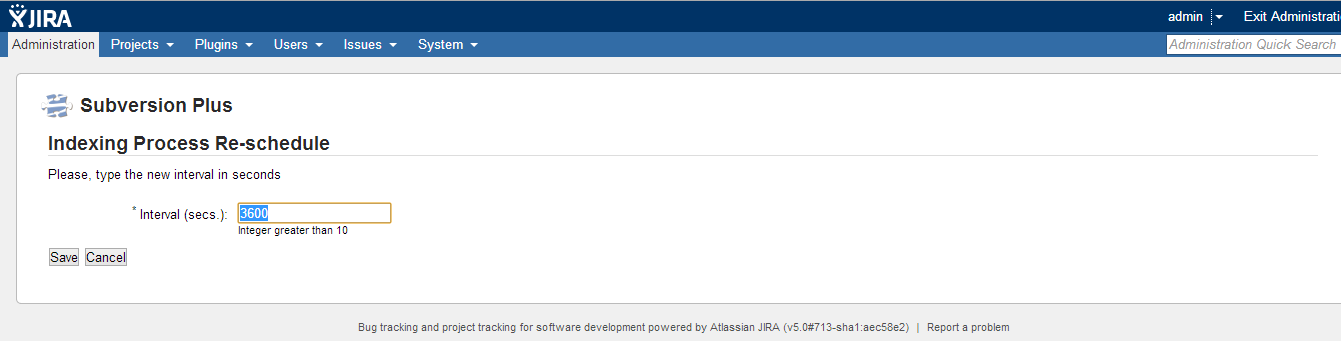
The registered repositories are periodically scanned in order to index the new commits. Subversion ALM will scan each repository incrementally, since its latest indexed commit. The maximum amount of parallel indexing repositories can be configured by changing the MAX\_INDEXING\_THREADS value at the APPLICATION\_PROPERTIES table on the internal database. The default value is 3.

The indexing interval can be scheduled from the *Indexing Process* section:



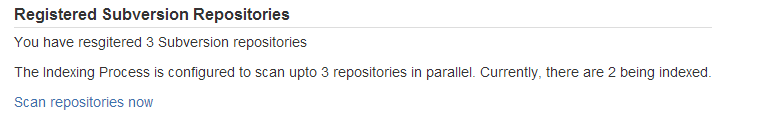
The default value is 3.600 seconds (1 hour) and the minimal supported value is 10 seconds.

The indexing interval can be modified by clicking on the *Re-schedule* link. This will open a new page:



The new interval value has to be in second units.

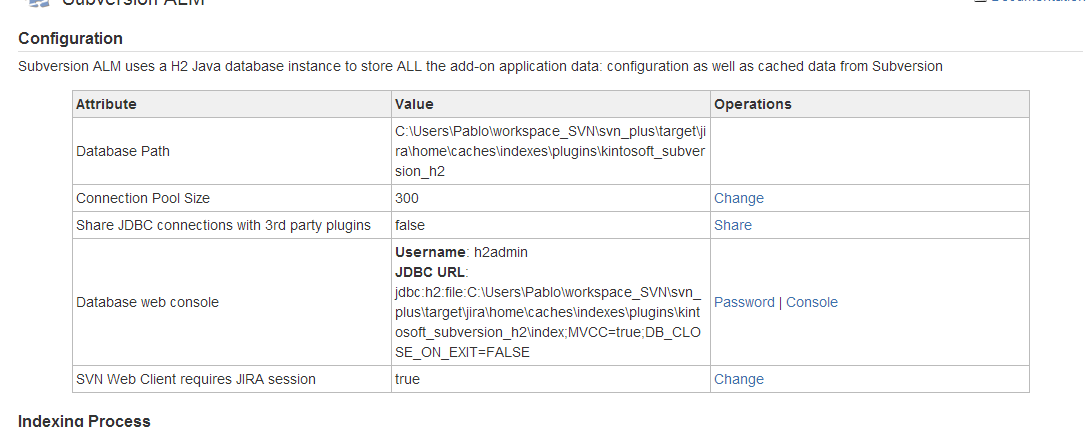
All the indexing information has been moved into the *Subversion Registered Repositories* section:



The *Scan repositories now* link is dynamic and it will displayed only when it has sense to start the Indexing Process manually without waiting for be fired by the scheduler

## Configuration parameters

All the application data, including the plugin’s own configuration, the repositories configuration as well as any data retrieved from Subversion, are stored in the H2 database instance. The database location is shown in *The Internal Database* section > *Database Path* entry.



### Re-initializing the Subversion ALM configuration

The H2 database instance can be deleted to fully reset Subversion ALM configuration:

1. Disable the plug-in
2. Delete the *Database* *Path* directory
3. Enable the plug-in

By following the steps above, a new clean database would be created. The entire previous configuration as well the Subversion indexed data will be permanently lost.

### Exporting and importing the registered repositories

Before deleting the database files, you might want to export all the registered repositories in order to import them after resetting the configuration.

#### Export to CVS

You have to export the following tables:

* REPOSITORIES
* REPOSITORIES\_CONFIGURATION

By running the SLQ commands below from the database web console:

* CALL CSVWRITE('repos.csv', 'SELECT \* FROM REPOSITORIES')
* CALL CSVWRITE('repos\_conf.csv', 'SELECT \* FROM REPOSITORIES\_CONFIGURATION')

**NOTE**: The Subversion credentials will be exported in plain text. Be careful with the repos.csv file and keep it in a SECURE location.

#### Import from CVS

Start the plug-in, this will create an empty database schema (tables, indexes, etc.). Then run:

* INSERT INTO REPOSITORIES (select \* from csvread('repos.csv'));
* INSERT INTO REPOSITORIES\_CONFIGURATION (select \* from csvread('repos\_conf.csv'));

And re-start the plug-in.

### The Subversion timeouts

Sometimes login into a Subversion repository or fetching the commits might take a long time. By default, the Connection and Read timeouts are two minutes (120000 milliseconds) but you might want to change them:

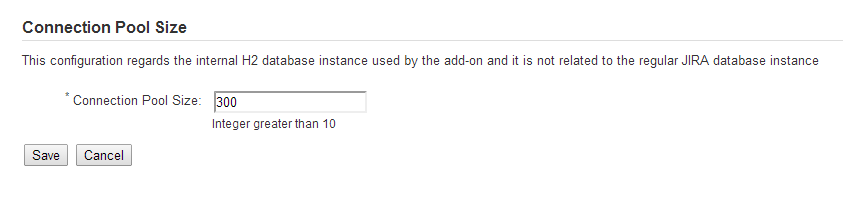


**NOTE**: The add-on must be re-started after modifying any timeout.

### The pool size

By default the plug-in creates 100 database connections during the start up. The code has also been written to ensure that each user action consumes one database connection only. If no free connections are available the add-on will wait for ten seconds before raise an error. If during that time interval some connection is released, then it will be assigned to some process (user action) waiting for a connection. Depending on your user base, it might require increasing the amount of available connections.

You can increase the *Connection Pool Size* value by clicking on the *Change* action:



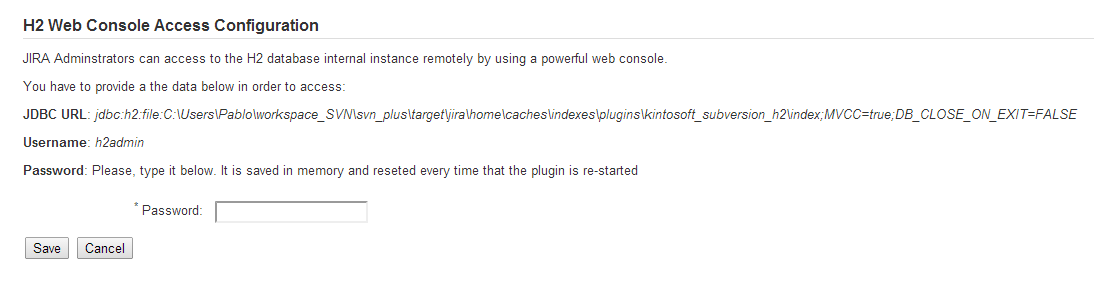
### The Database Web Console

The H2 database is automatically started and stopped by Subversion ALM. It is opened in the *embedded* mode and the database file is locked while Subversion ALM is enabled: no any external process can access/open the H2 database.

The database can be accessed remotely through the *Database Web Console*. It requires JIRA Administer privileges.

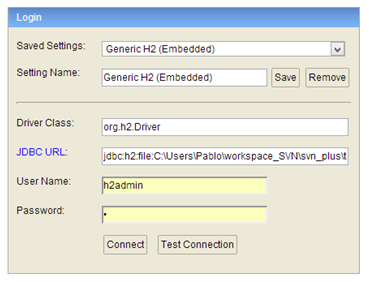
The Database Web Console allows to explore the data model (tables) as well as read/modify any data. Please, use it with caution.

Prior to access to the web console, setting a password is required. Click on the *Password* action and provide a valid one:



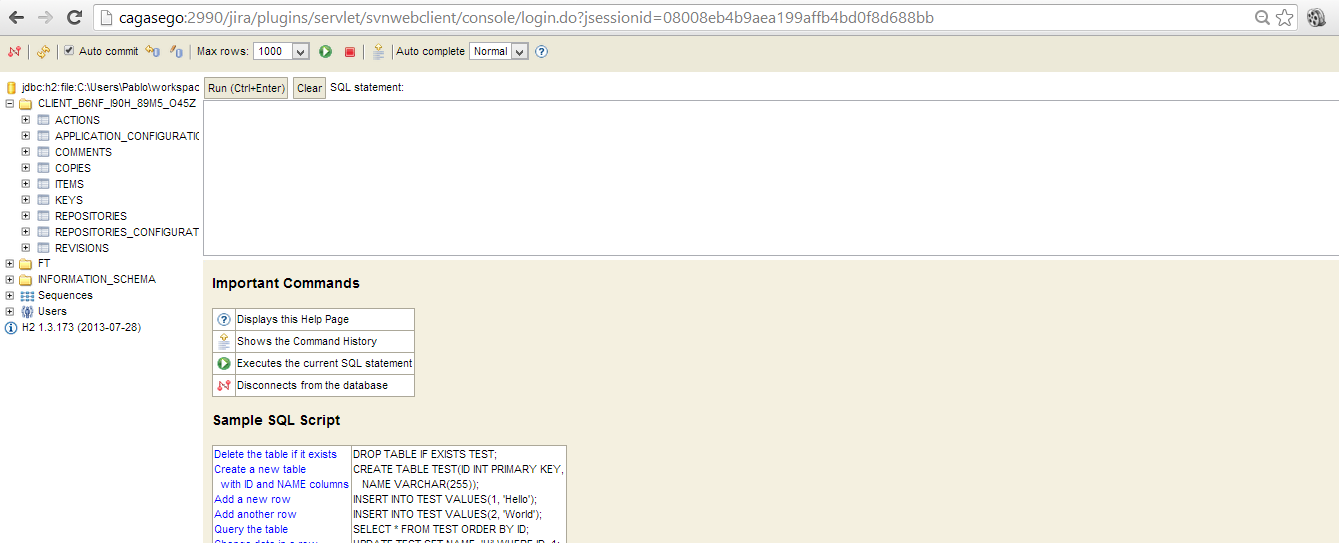
Then save it and copy the JDBC URL into the system clipboard because it will be required in next steps. Also please pay attention to the *h2admin* username.

JIRA administrators can click on the *Console* link to open the Web Console:



And fill out the required connection data:

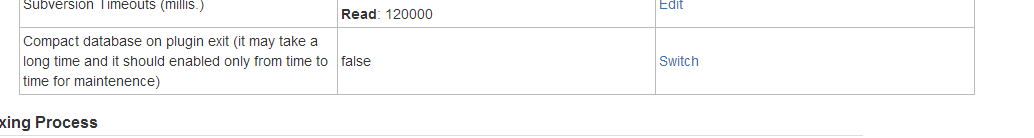
Type the *JDBC URL* from and the user credentials: fill the *User Name* with the *h2admin* default value and also fill the *Password* field with the value that you introduced in the previous steps. Click on the *Connect* button to access to the H2 database:



The console allows exploring the database data and **modifying, inserting and deleting** new records. It is really powerful as you can use it to fully administer your H2 database instance.

### Compacting the database file size

A new parameter is supported to allow compacting the database file size on database shutdown (upgrading, disabling and deleting repositories):

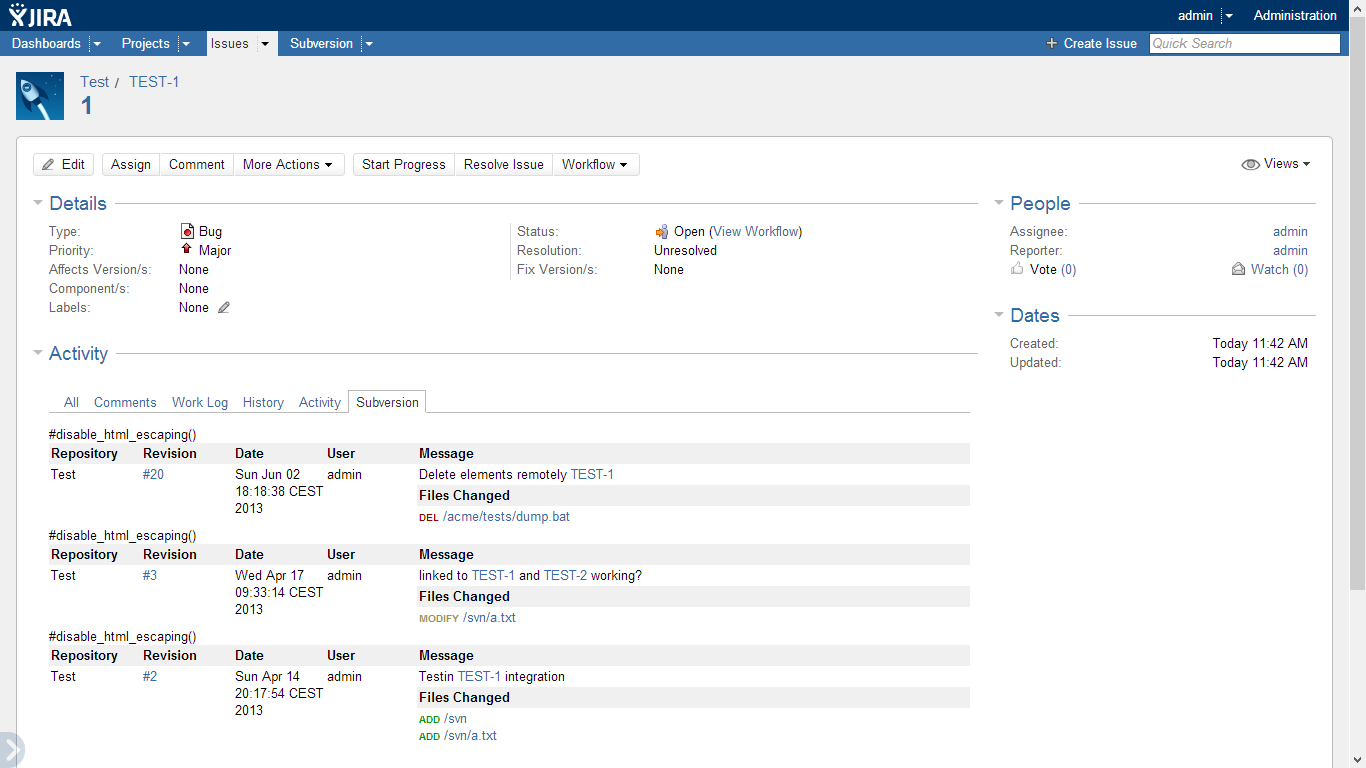


The default value is *false* and usually it is not necessary compacting the database files. However, you may want to enable it from time to time and set it to false (default value) after re-enabling the plugin as compacting rebuilds the database indexes and it could take a long time.

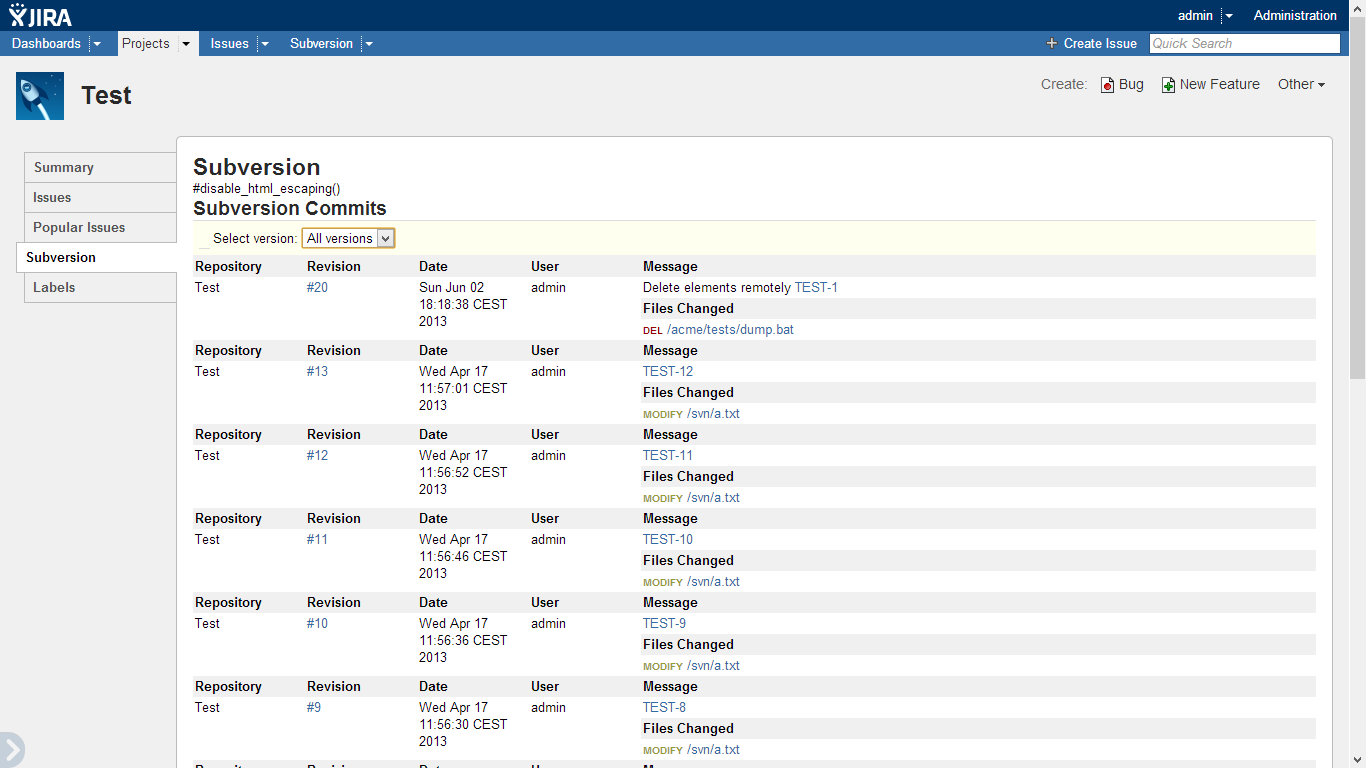
# Displaying Subversion commits on JIRA

The indexing process scans the repositories looking for JIRA issue keys embedded in commit messages and stores those relationships in the H2 database. They can be explored from the JIRA from two locations:

1. *The issue page*: Subversion ALM adds a new *Subversion* tab in the JIRA issue page. It lists the commits related to the issue:



1. *The project page*: It also adds a *Subversion* tab on the JIRA project page:



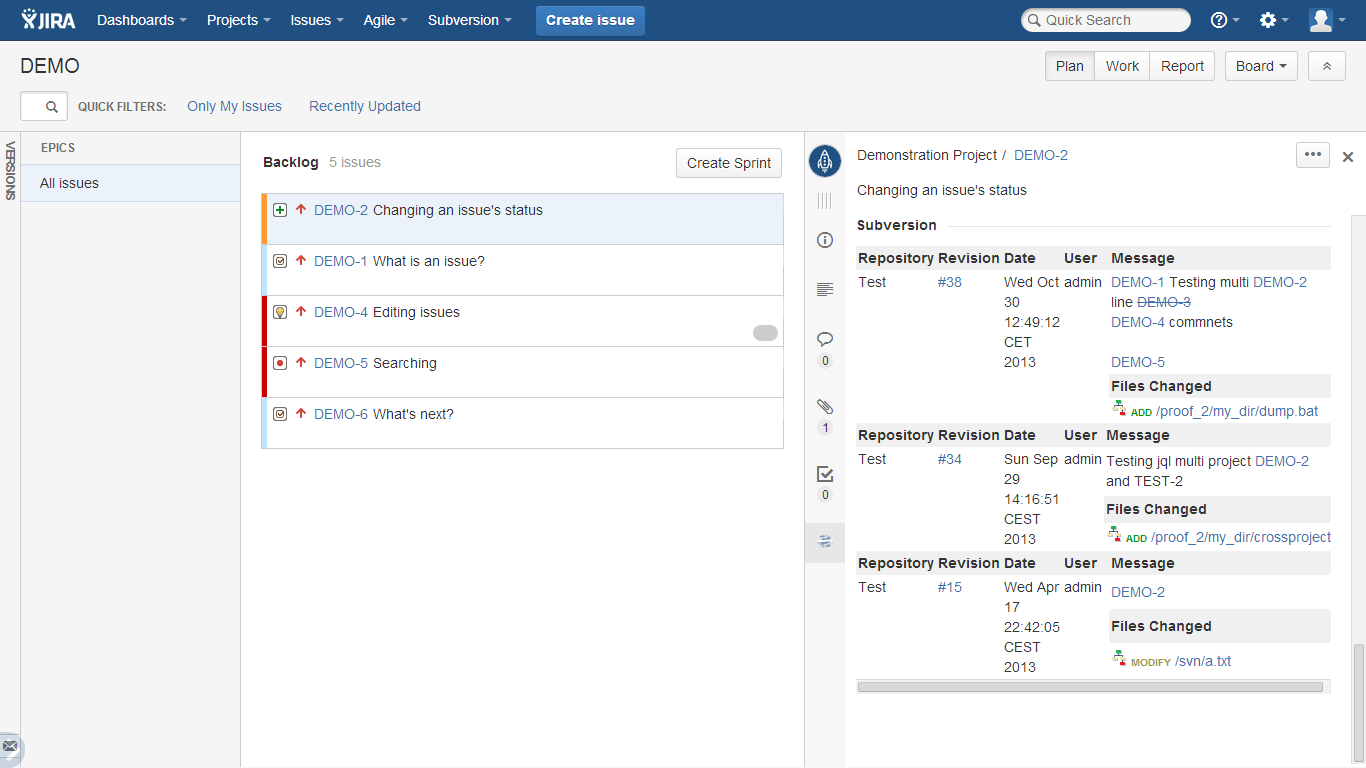
In this case, a combo box is displayed at the top of the view allowing users to filter by project *version*.

**IMPORTANT**: Those views require that the user is granted with the **View Development Tools**, ***View Issue Source Tab*** or ***View Version Control*** permission accordingly to your JIRA version.

**IMPORTANT2**: The permissions above have been discontinued in JIRA 7. As there is no chance to verify them, the add-on ignores them.

# Displaying Subversion commits on Agile (Greenhopper)

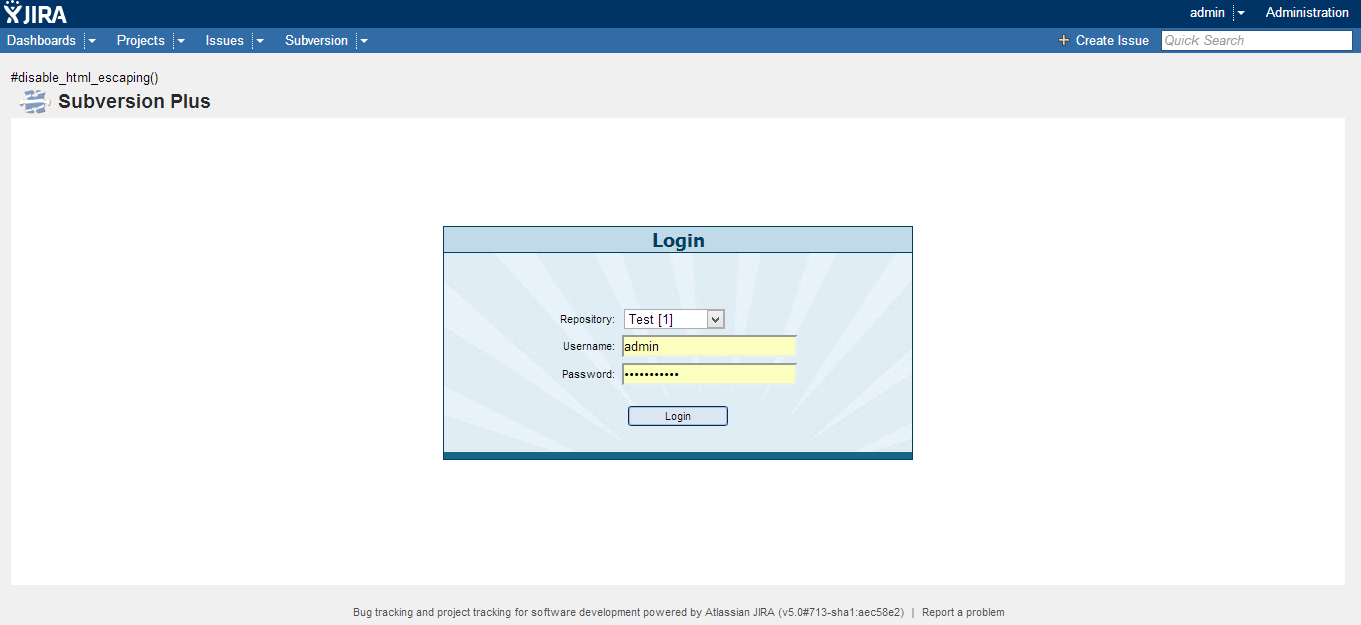
Since the 5.8.0 version, the 10 latest commits are displayed on the JIR Agile Scrum Plan and Kanban Work boards:



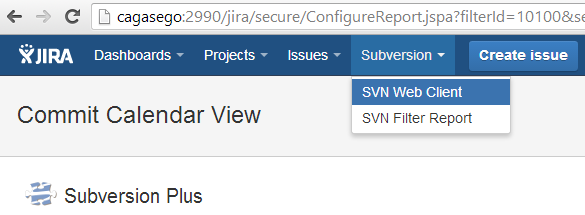
# The Web Client for Subversion

## JIRA integration

Since the 2.0 version, a customized version of the *Polarion Subversion Web Client* is bundled with the plug-in. It is automatically deployed and configured by Subversion ALM to *work* fully integrated with JIRA *out-of-the-box*:

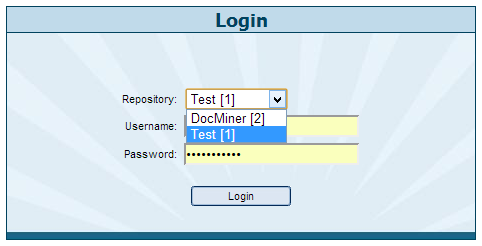


The Web Client is a full web application and it can be accessed from the Subversion menu on the JIRA top action bar:



##### The *Subversion* top menu is displayed if the user has the enough privileges (just the same required to show the Subversion tab on the issue and project pages).

The users can log out from the Subversion Web Client at any time by clicking on the *logout* link at the top right of the view. It will show the login view and the users can choose any registered repository from the *Repository* combo box:

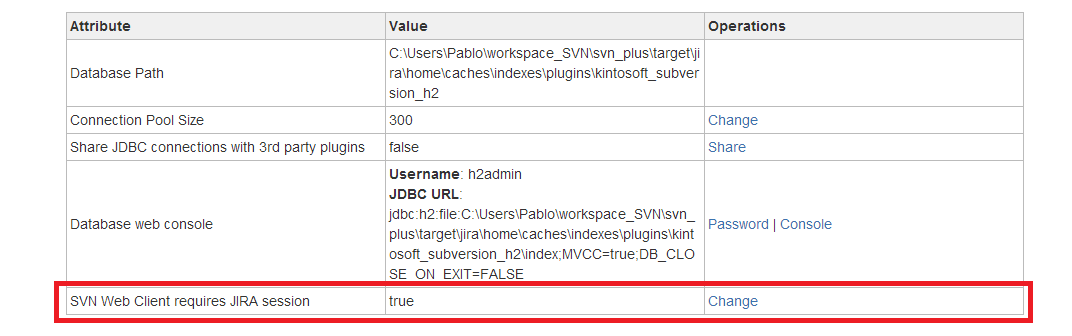


The combo box is populated by Subversion ALM with the names of all the registered repositories.

**The repository Id is shown beside the repository name**. This is very useful in order to use the JQL functions as many of them required the repository Id as input parameter.

## Subversion Web Client in *Standalone* mode

By default, all the users have to login JIRA prior browsing on Subversion. This behavior can be changed from the plugin configuration view:

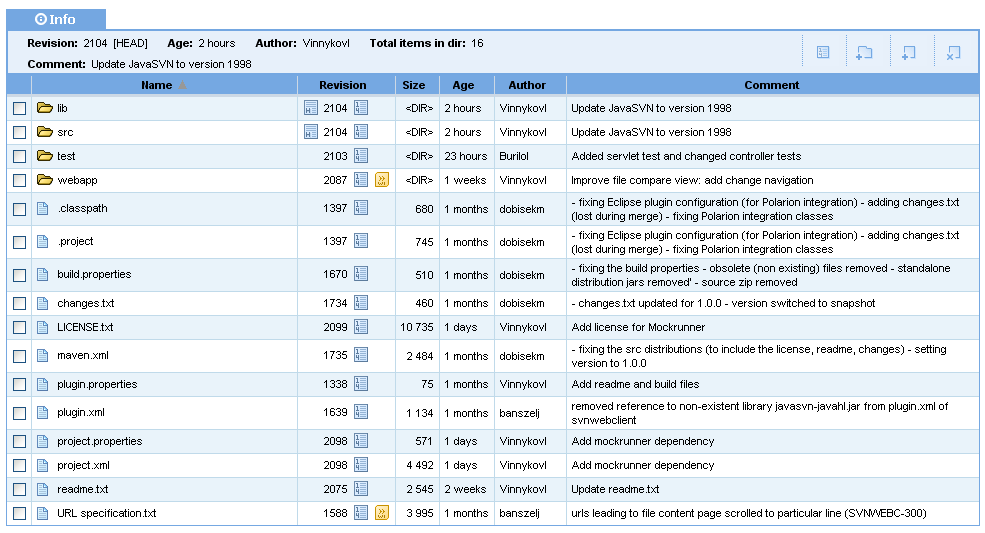


If the *SVN Web Client requires JIRA Session* attribute is set to *false,* then the plug-in does not validate any JIRA session allowing browsing on Subversion to any user logged in Subversion. In this mode, Subversion ALM can be used as a pure Subversion web client and users have not be aware about JIRA. The Subversion Web Client can be accessed from the following URL directly:

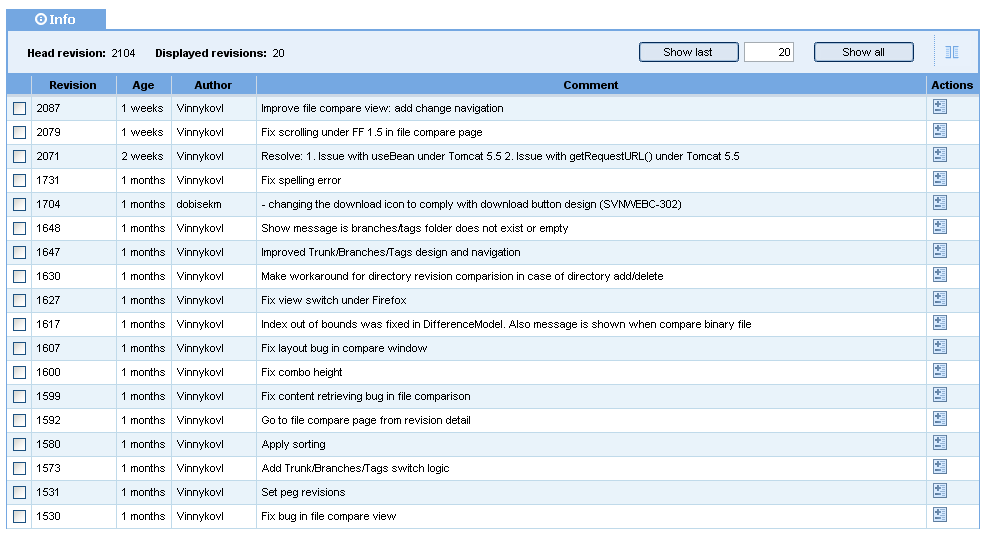
*<your JIRA base URL>****/plugins/servlet/svnwebclient/login.jsp***

## Major features

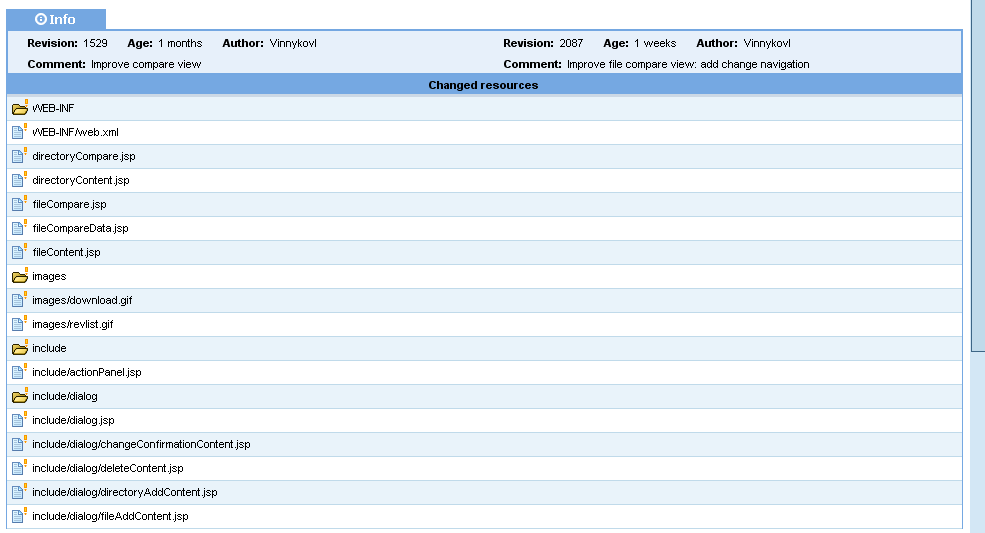
### Repository browse



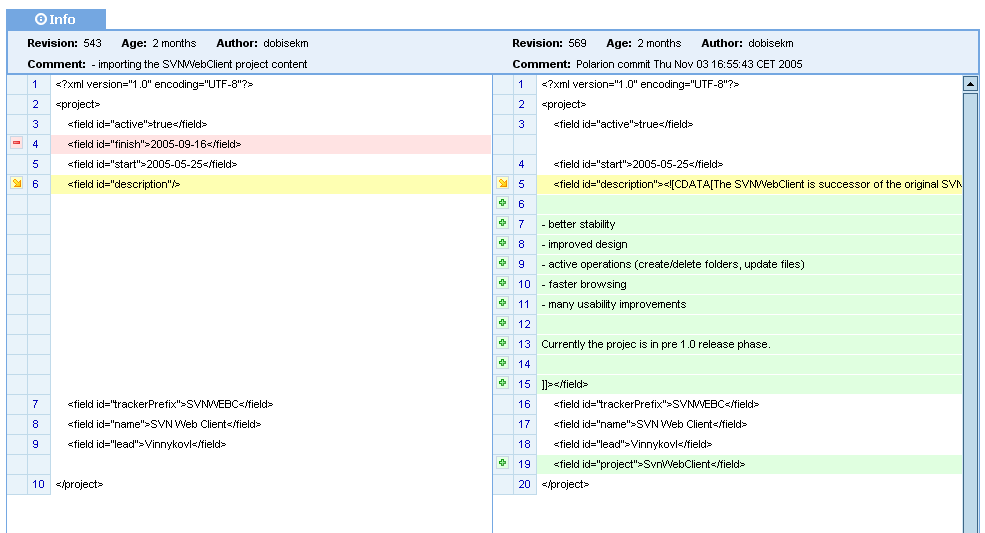
### Revision List



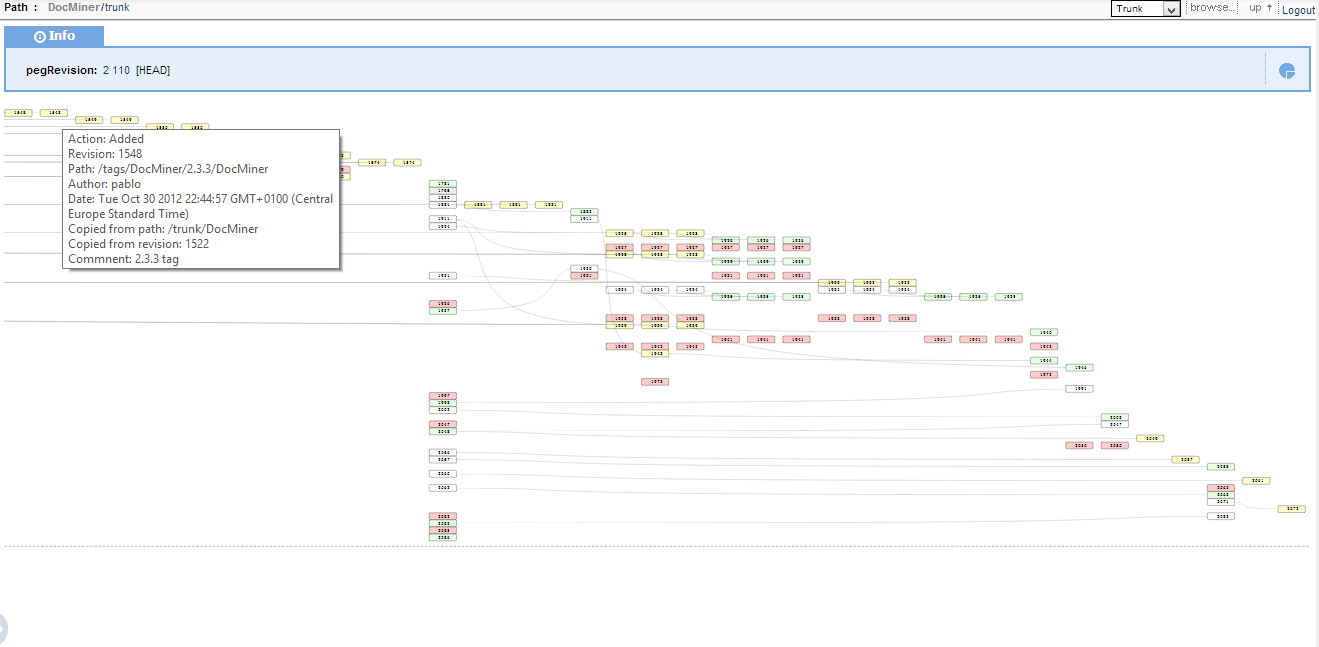
### Revision changes



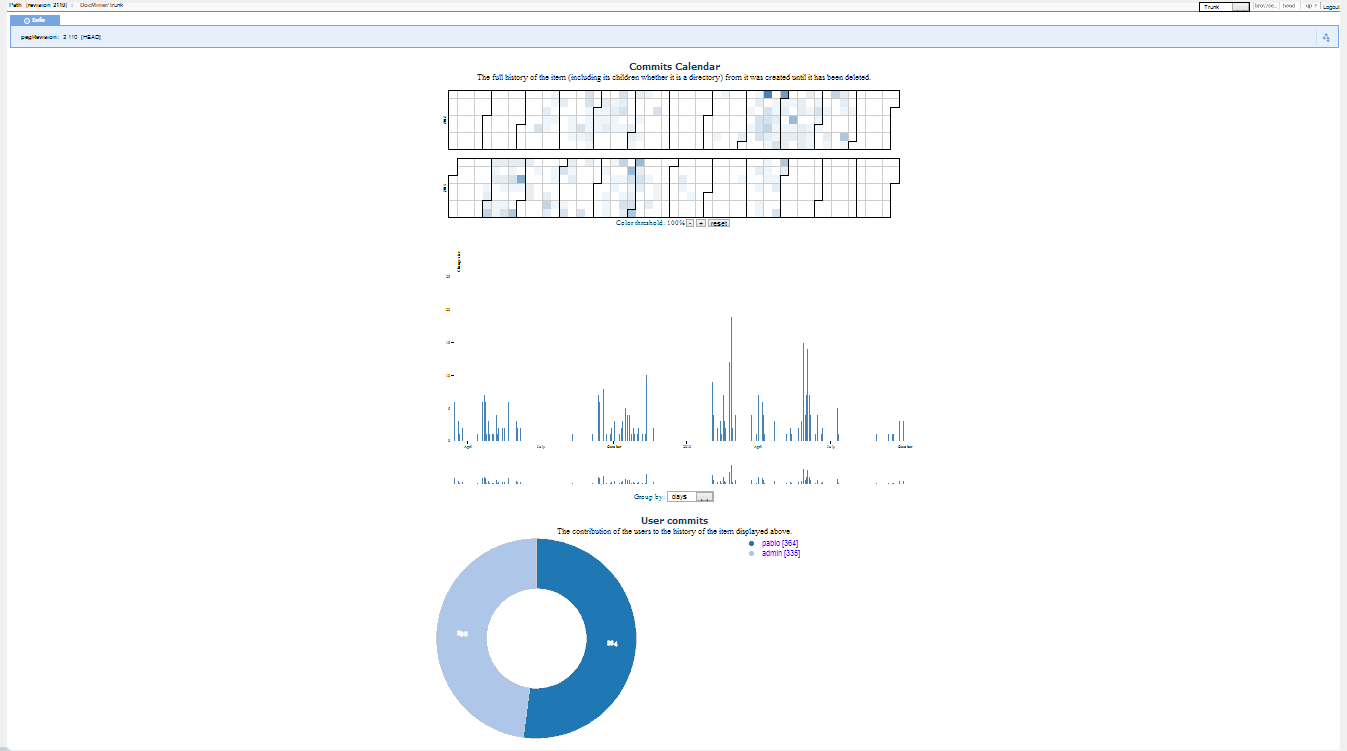
### Compare files



### Commit graphs



### Statistics



# Reporting

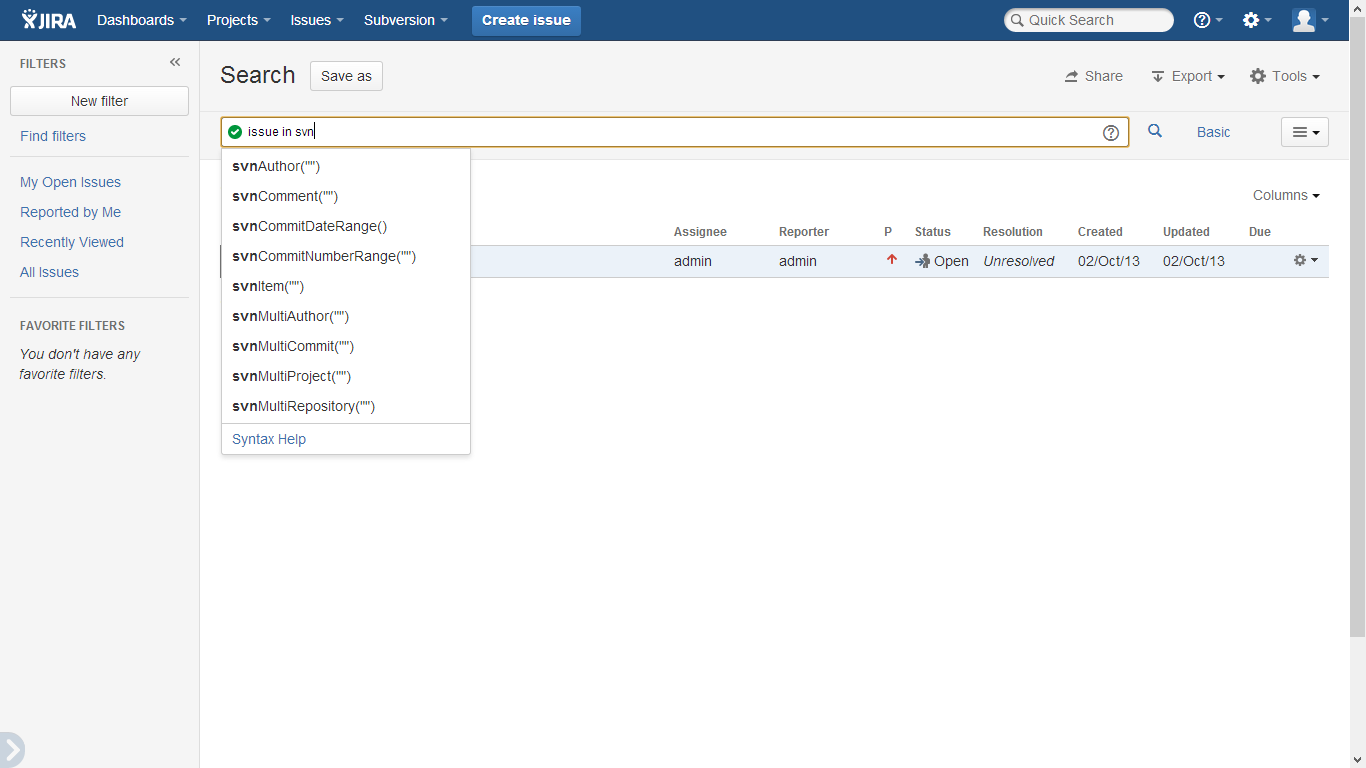
Subversion ALM supports a very powerful and flexible reporting based on:

* JQL
* Commit Calendar View

The JQL functions provided by Subversion ALM allows searching issues filtering by Subversion attributes like author, date range, commit range, words in comment, files and subdirectories, etc. while the Commit Calendar View shows commits in an interactive calendar.

## JQL functions

Several JQL functions are supported by Subversion ALM allowing users to search issues on JIRA by using Subversion attributes like the *commit author, commit date range, committed files and directories, any words in the commit message*, etc.



All the Subversion ALM JQL functions:

* Start with the **svn** prefix.
* Show a useful quick guide on JIRA when some parameter has an invalid value.
* Support the following two parameters:

1. *repoId*: it restricts the search the repository with the provided Id (Id>0). An Id with the zero value always means all the repositories.
2. *limit:*  It is very important understanding that the JIRA JQL engine always re-orders the JQL results by some Issue attribute (priority, status, …). Therefore, is not possible displaying on JIRA the issues sorted by any Subversion attribute like the revision number, commit date, etc. To deal with this, when the *limit* filter is set, the results are internally sorted by the *commit day* before they are sent back to JIRA. A limit = 10 means the 10 issues related to the latest (newer) commits matching the query, but they will be re-ordered by JIRA when they are displayed. Regardless they are re-sorted by JIRA you can be confident about they are the most recent issues.

To quick start working with the Subversion ALM JQL functions type the query below in the JIRA Issue Advanced Search:

***issue in svn***

Currently the Subversion ALM JQL functions are grouped into two main groups:

* *Attribute functions*: they allow searching issues on JIRA by using Subversion attributes:
  + ***svnCommitNumberRange***
  + ***svnCommitDateRange***
  + ***svnAuthor***
  + ***svnComment***
  + ***svnItem***
* *Aggregation functions*: they allow searching issues on JIRA by aggregating (counting/amount) some Subversion attribute:
  + ***svnMultiCommit***
  + ***svnMultiAuthor***
  + ***svnMultiProject***
  + ***svnMultiRepository***

### Attribute JQL functions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnCommitNumberRange*** | | | | | |
| It returns the issues related to commits between a revision range on an specific repository | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** | **Description** |
| 1 | repoId | Integer | Yes | >0 | The repository Id |
| 2 | start | Integer | No | >=0 | The start revision of the range |
| 3 | end | Integer | No | >= 0 | The end revision of the range |
| 4 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnCommitNumberRange****(5,4364,””,10)*

Meaning:

It returns maximum 10 issues related to the latest commits from the 4364 (included) revision to the HEAD in the repository Id=5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnCommitDateRange*** | | | | | |
| It returns the issues related to commits between a date range on all the repositories | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** | **Description** |
| 1 | repoId | Integer | Yes | >=0 | Zero means all the repositories. |
| 2 | start | String | No | yyyy-MM-dd | The start date of the range |
| 3 | end | String | No | yyyy-MM-dd | The end date of the range |
| 4 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnCommitDateRange*** *(0,”2013-02-01”,””,99)*

It returns maximum 99 issues related to the latest commits on any repository from the February, 1st 2013.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnAuthor*** | | | | | |
| It returns the issues related to commits made for an specific user | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** | **Description** |
| 1 | username | String | No | Any | An empty string means no username (null). |
| 2 | repoId | Integer | No | >=0 | Zero means all the repositories. |
| 3 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnAuthor*** *(”sally”,0,1)*

It returns one issue (if any) related to the latest Sally’s commit on any repository.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnComment*** | | | | | |
| It returns issues related to commits containing all the exact words in their commit comment messages | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** | **Description** |
| 1 | Lucene Query | String | Yes | Any | A Lucene Query |
| 2 | repoId | Integer | No | >=0 | Zero means all the repositories. |
| 3 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnComment*** *(”merge\* AND r1034”, 3)*

It returns all the issues related to any commit (without any specific order) matching the Lucene query for the repository Id=3. In this example “merg**ed** in revision r1034” would match the query

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnItem*** | | | | | |
| It returns the issues related to commits modifying a file or subdirectory | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** |  |
| 1 | Item | String | Yes | Path | A path or file name |
| 2 | Action | String | No | A,M,D,R,”” | The action on the item |
| 3 | repoId | Integer | No | >=0 | Zero means all the repositories. |
| 4 | limit | Integer | No | >=1 | Limits the number of returned issues |

Examples:

1. *issue in* ***svnItem****(“myfile.txt”,”D”,3)*
2. *issue in* ***svnItem****(“/foo/myfile.txt”,””,3,5)*
3. *issue in* ***svnItem****(“/foo”,”A”,3)*
4. *issue in* ***svnItem****(“/foo/”,””,3,10)*
5. It returns all the issues related to the commit which deleted a “myfile.txt” file in the repository 3 regardless the file location. As no path is set, it will search for the file in all the branches.
6. It returns the issues related to the latest 5 commits changing the exact “/foo/myfile.text” file in the repository 3. As the path is set to “/foo” it will look for the file in that exact branch.
7. It returns all the issues related to the commit which created the exact “/foo” branch. Notice that the branch name does not end with a slash “/” character, so the subdirectories are ignored.
8. It returns 10 issues related to all the commits modifying something under the “/foo” branch. Notice that the branch name ends with a slash “/”, hence all the subdirectories will be taken in consideration during the search.

### Aggregation JQL functions

Sometimes managers want to track some potential risks. Subversion ALM supports some useful JQL functions in order to help them:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnMultiCommit*** | | | | | |
| It returns issues related to a minimal amount commits | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** |  |
| 1 | count | Integer | Yes | > 0 | Minimal amount of commits |
| 2 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnMultiCommit*** *(5, 10)*

What are the most recent 10 issues which have been related to 5 or more commits on any repository?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnMultiAuthor*** | | | | | |
| It returns issues with commits from several authors | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** |  |
| 1 | count | Integer | Yes | > 0 | Minimal amount of different authors |
| 2 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnMultiAuthor*** *(3, 10)*

What are the 10 most recent issues which have commits from more than 3 different users?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnMultiRepository*** | | | | | |
| It returns issues with commits on different repositories | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** |  |
| 1 | count | Integer | Yes | > 0 | Minimal amount of different repositories |
| 2 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

issue in ***svnMultiRepository*** *(2)*

What are all the issues having commits on 2 or more repositories? (Cross repository issues are not allowed in this company).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***svnMultiProject*** | | | | | |
| It returns issues with commits affecting to different JIRA projects | | | | | |
| **#** | **Name** | **Type** | **Required** | **Values** |  |
| 1 | commits | Integer | Yes | > 0 | A path or file name |
| 2 | limit | Integer | No | >=1 | Limits the number of returned issues |

Example:

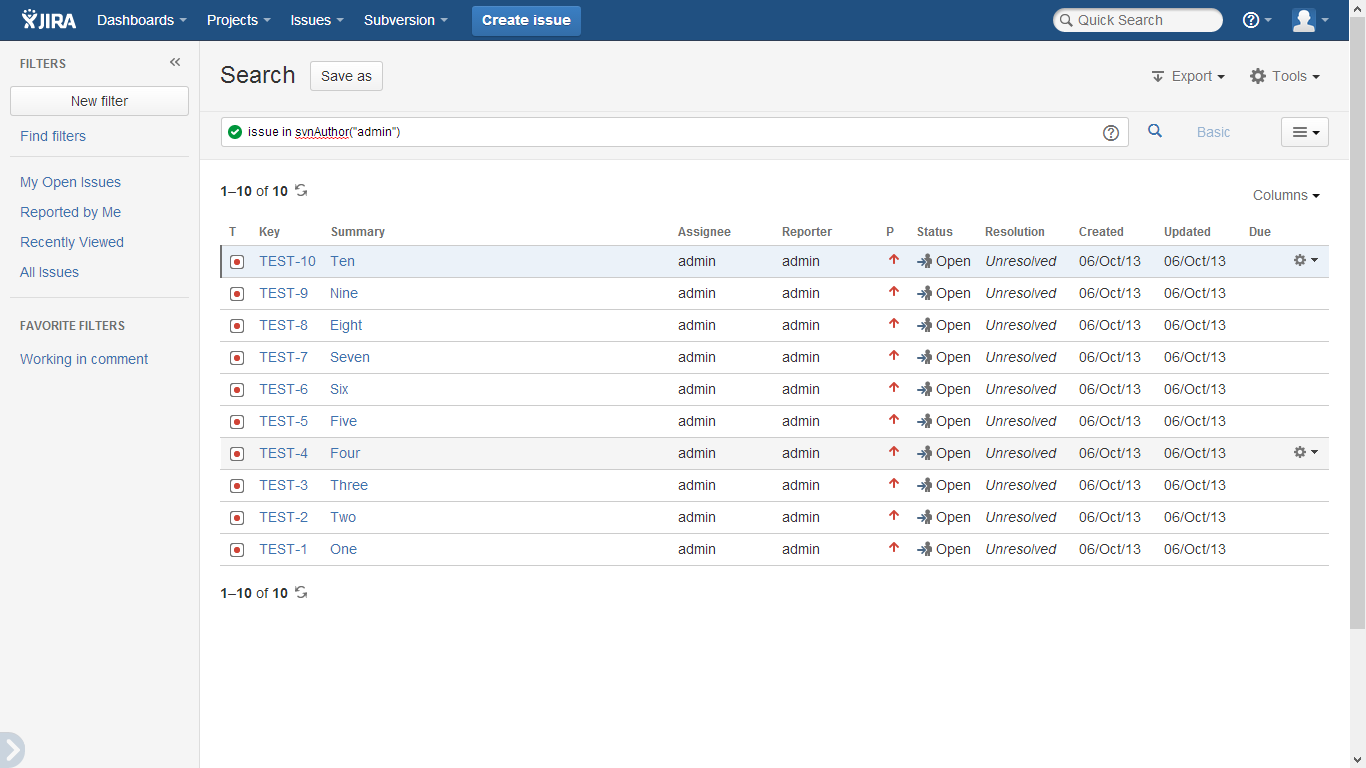
issue in ***svnMultiProject*** *(2, 10)*

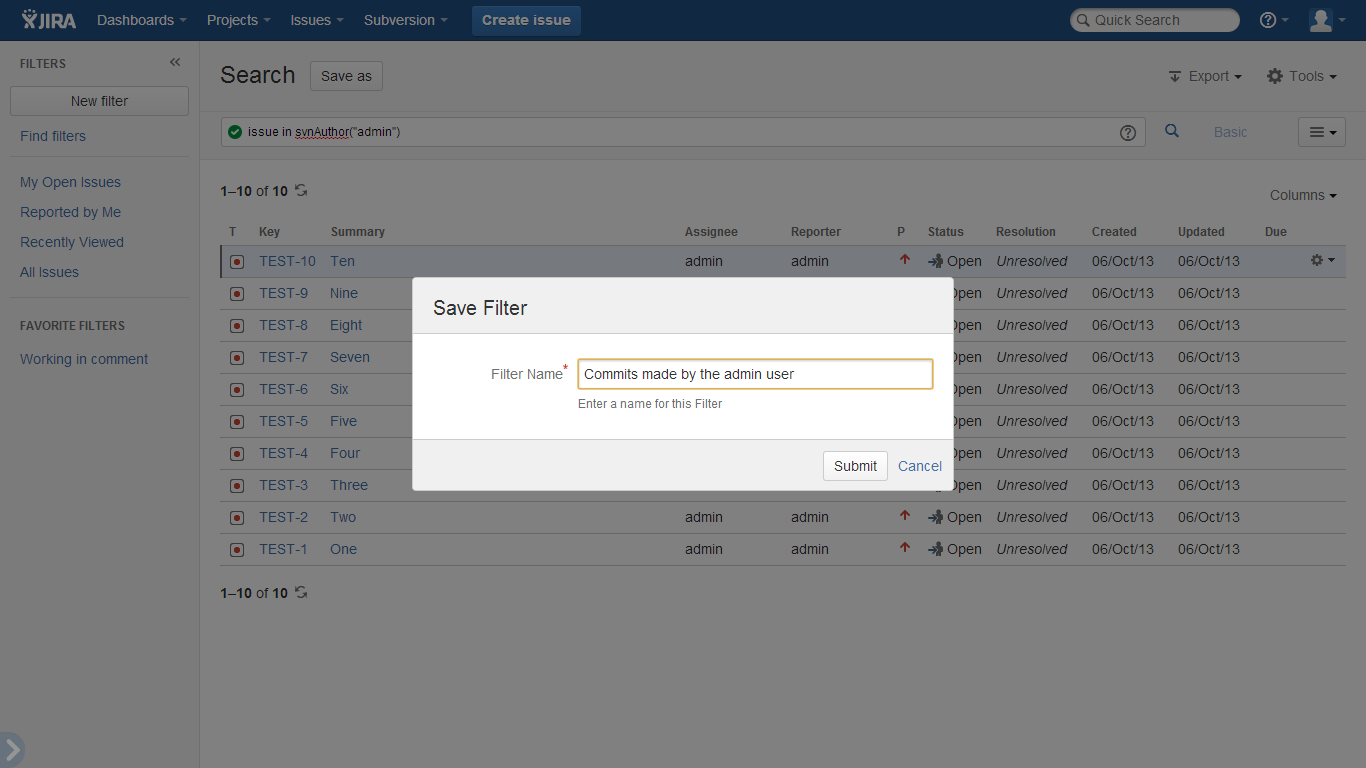
What are the recent issues that have commits on 2 on more JIRA projects?

## Commits Calendar Report

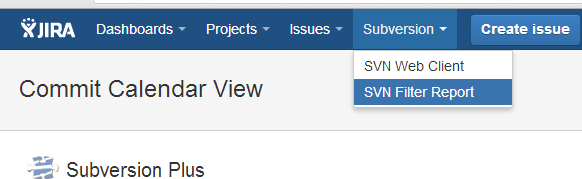
The Commits Calendar Report allows visualizing the commits related to the issues returned by any filter.

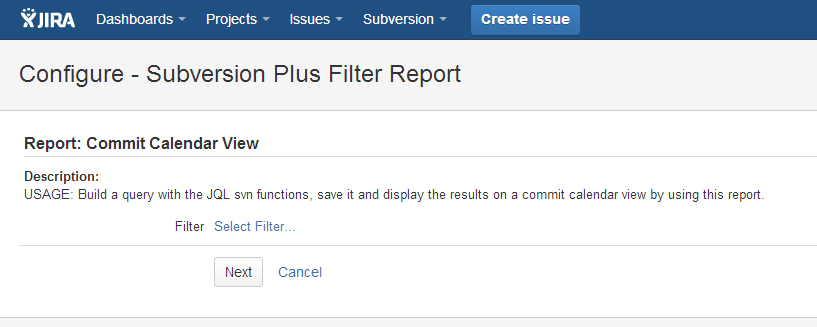
Prior to use it, a JQL query has to be saved:



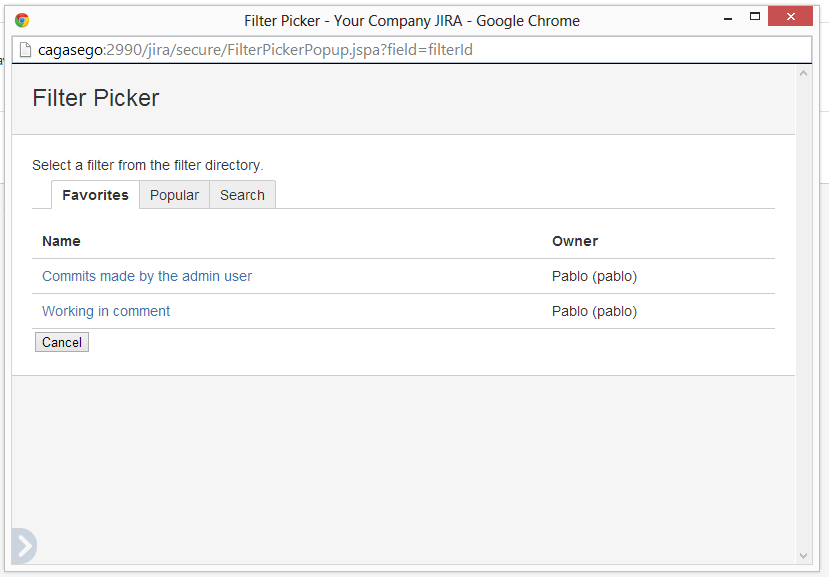


From the top menu: *Subversion* > *SVN Filter Report*

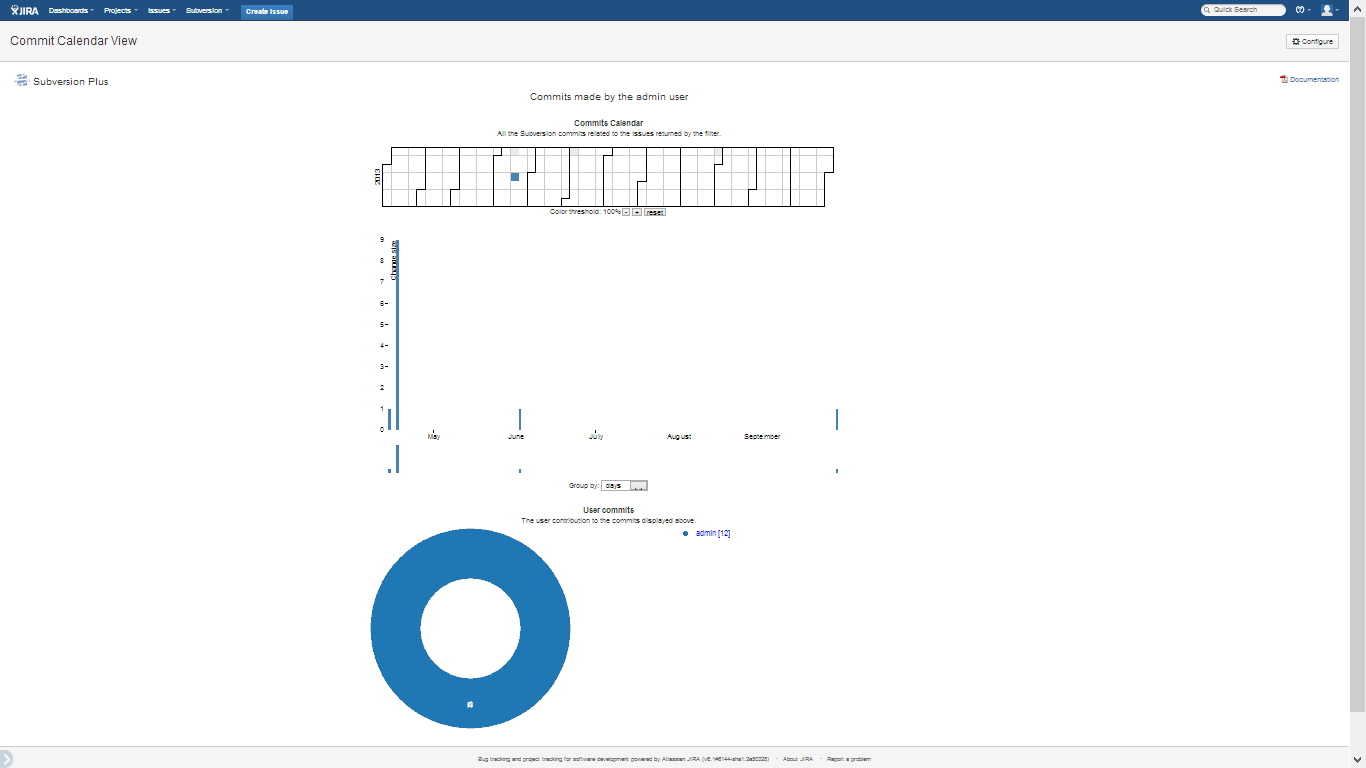




Click on the *Select Filter…* link:



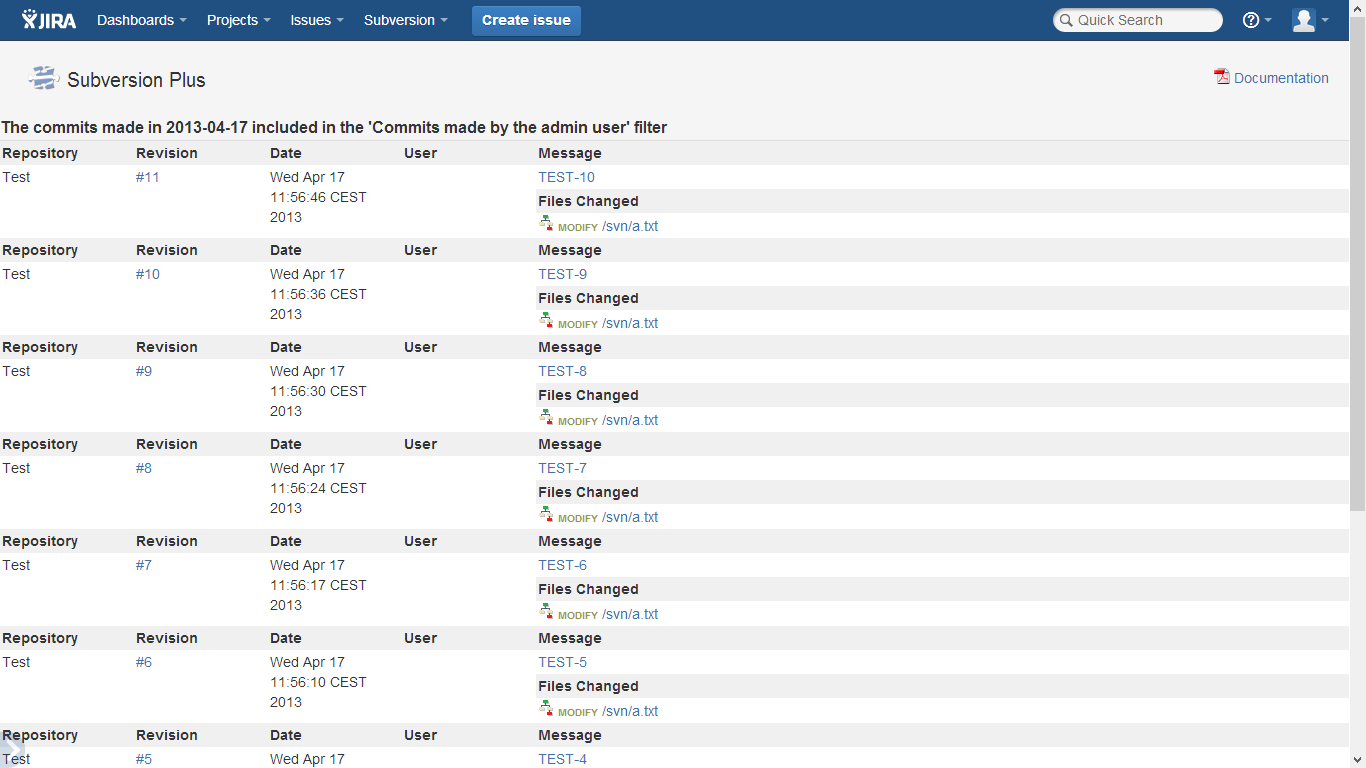
Select a filter and click on the *Next* button:



This page shows:

* a *calendar* at the top of the page displaying the commits related to the JIRA issues along the time
* a *bar chart* at the middle showing the amount of commits vs time. By default the commits are displayed grouped by day, although they can be re-grouped in weeks: there is an interactive thumbnail view below allowing to select a range of days.
* a *pie chart* at the bottom with the contribution of each author.

You can click on any day on the top calendar as well as on the authors’ names links in the pie chart to see the issues on JIRA:



# Extending the functionality with 3rd party plugins

Since the 5.5 version, sharing the database connection among 3rd party plug-ins is supported. And since the 5.8 version it publishes commit events allowing subscription.

## How to use the JDBC connections from a 3rd party plug-in

This brings a lot of power to users and allows them to enhance the current functionality by creating custom plug-ins in order to:

* Writing a GUI supporting to change JIRA issue keys related to commits when users forget or type a wrong key related to a commit
* Capture JIRA moved issue events to keep aligned the commits
* Build powerful reports
* ...

1. Add a copy of the "SWCPublicInterface" Java interface into your sources plugin:

com.kintosoft.svnwebclient.jira.public\_interface.SWCPublicInterface

the interface above provides a "getConnection()" method returning an standard JDBC connection

2. Import the implementation of the interface provided by the Subversion ALM plugin

by adding the component-import element below into your atlassian-plugin.xml descriptor:

<component-import key="svnalm-public">

<interface>

com.kintosoft.svnwebclient.jira.public\_interface.SWCPublicInterface

</interface>

</component-import>

3. Import the "com.kintosoft.svnwebclient.jira.public\_interface" from your Maven *pom.xml* file to make it visible from your plugin:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<build>

<plugins>

<plugin>

<configuration>

<instructions>

<Import-Package>

com.kintosoft.svnwebclient.jira.public\_interface\*;version="0.0.0"

</Import-Package>

....

4. Use injection in the constructor to get an instance of the:

public class Foo{

final private SWCPublicInterface swci = null;

public Foo(SWCPublicInterface swci){

this.swci = swci;

}

}

4. Get a JDBC connection and do not forget make a commit/rollback and CLOSE THE CONNECTION in

order to return it to the pool:

Connection conn = null;

PreparedStatement ps = null;

try {

try {

int commits = 0;

conn = swc.getConnection();

String sql = "select count(\*) as count from KEYS where project=?";

ps = conn.prepareStatement(sql);

ps.setString(1, key);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

commits = rs.getInt("count");

}

rs.close();

startingParams.put("commits", commits);

} finally {

if (ps != null) {

ps.close();

}

if (conn != null) {

conn.rollback(); // rollback or commit accordingly

conn.close(); // return the connection to the pool

}

}

} catch (SQLException ex) {

System.out.println(ex.getMessage());

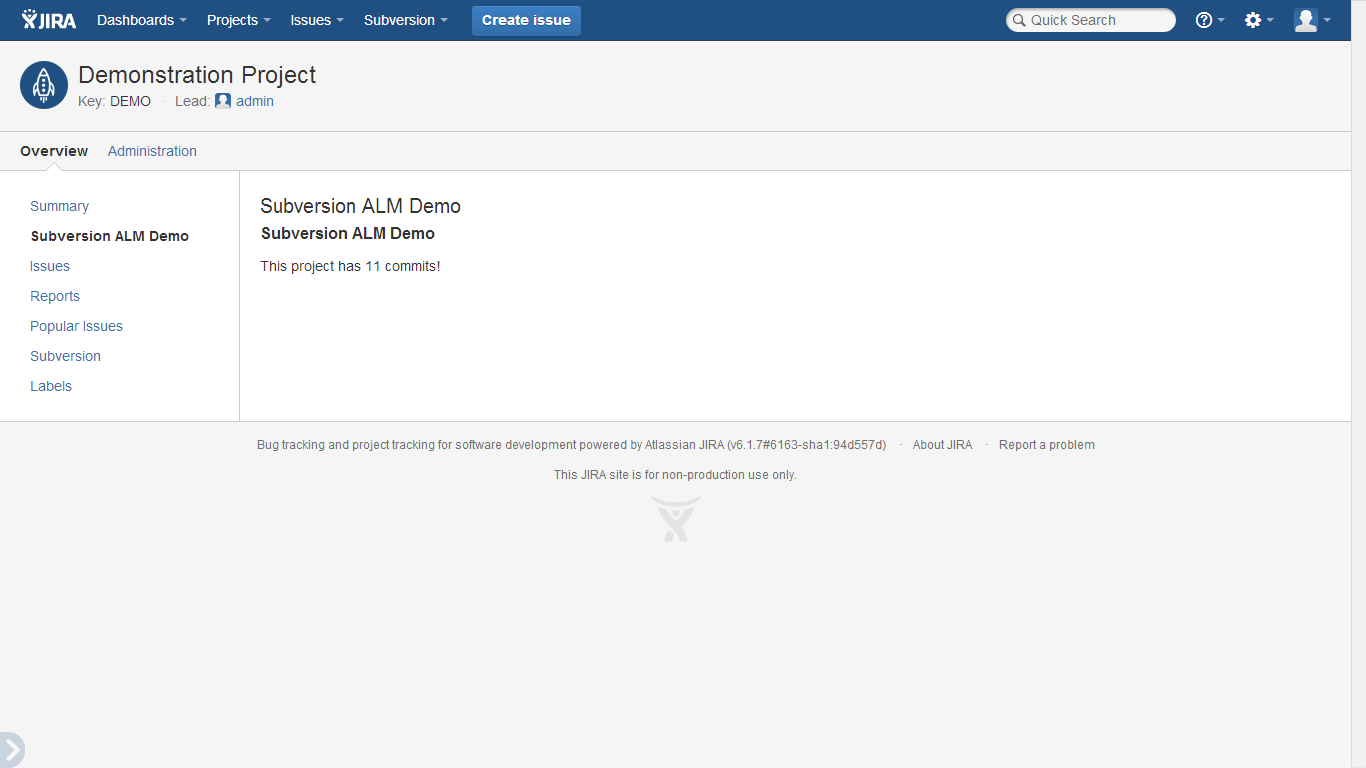
}

### Example

There is available an open source public example available on Google Code released under the Apache license:

<https://code.google.com/p/jira-svn-alm-db-demo/>

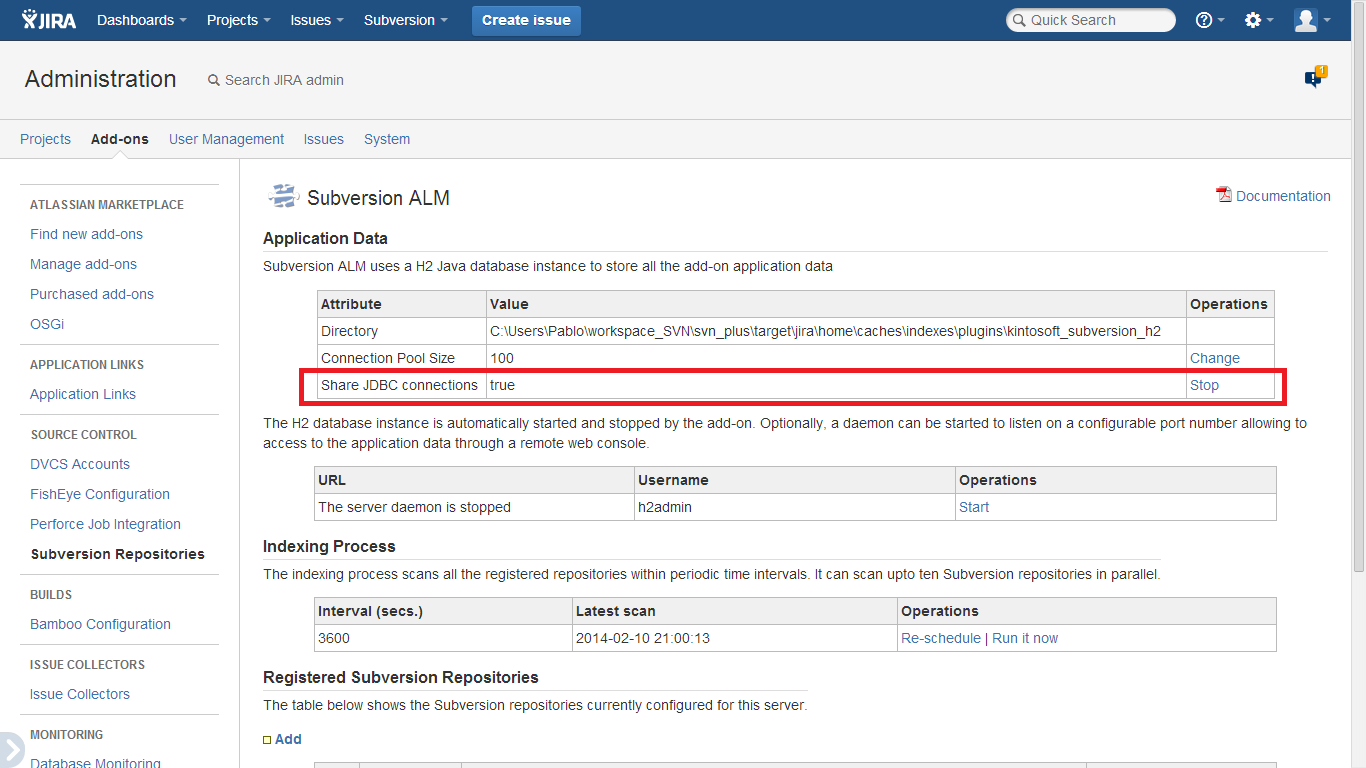
This example creates a *Subversion ALM Demo* tab on JIRA projects and shows the amount of commits:



### Security

Exposing the JDBC connection brings potential security risks as a malicious 3rd party plug-in could gain access to the Subversion ALM database and get the passwords of all the registered Subversion repositories!

This would be pretty difficult as all the plugins on the Marketplace are validated by Atlassian. However, to bring some peace of mind to JIRA administrators, a new feature was introduced in the 5.5.1 version: Sharing the JDBC connection must be explicitly enabled by a JIRA administrator:

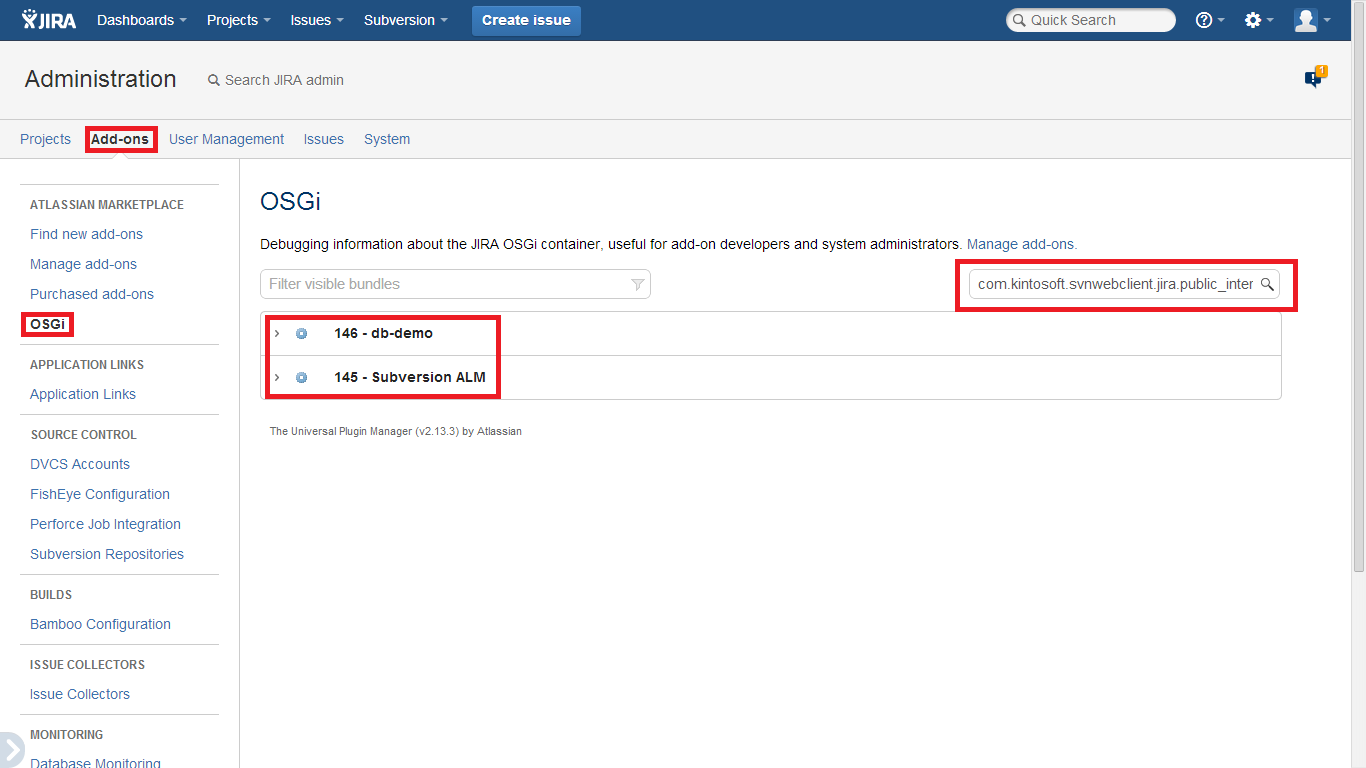


Before sharing any connection, the Subversion ALM plug-in checks the *Share JDBC connection* value. If it is false, then an exception is raised aborting the process and nobody can get a JDBC connection.

### Audit

How to know what plug-ins are accessing to the Subversion ALM internal database?

All the plug-ins have to explicitly import the com.kintosoft.svnwebclient.jira.public\_interface Java package in order to access to the Java object serving JDBC connections. Fortunately, JIRA tracks such information and allows administrators to check who is accessing to those objects from the OSGi console:



Copy the package name and paste it into the box at the top right. This will unveil the add-ons using the Subversion JDBC connection ☺

## Subscribing to Commit Events

Commits Events is a powerful feature supported from the 5.7.0 version. It allows to 3rd party plug-ins listen to all the commits fetched from Subversion and extend the functionality to create review issues or Smart Commits among other features.

In order to implement your own solution you have to follow the steps below:

1. Add a dependency into your pom.xml for the Google's Guava jar library. This library is provided by the JIRA System plugin and (among other features)

it provides the Eventbus component for event publish/subscription:

<dependency>

<groupId>com.google.guava</groupId>

<artifactId>guava</artifactId>

<version>10.0.1</version>

<scope>provided</scope>

</dependency>

2. Import the *svnalm-commits-eventbus* component shared by Subversion ALM.

<component-import key="svnalm-commits-eventbus">

<interface>com.kintosoft.svnwebclient.jira.public\_interface.CommitsEventbusService

</interface>

</component-import>

3. Add a copy of the *com.kintosoft.svnwebclient.jira.public\_interface.CommitEvent* class into your plug-in. Subversion ALM creates and dispatches a new *CommitEvent* object for each commit fetched from Subversion. This class supports the repositoryId, the commit number (revision) and the timestamp with the milliseconds when the *CommitEvent* was fired.

4. Create your own listener (subscriber) class. Have a look at the *com.kintosoft.jira.svn.MyPluginComponentImpl.CommitEventListener* class for an example.

Accordingly to the Eventbus specification, you have to add a method with the *@Subscribe* annotation for *CommitEvents* handling:

@Subscribe

public void handler(CommitEvent event) {

log.info(event.getRevision() + " revision commited in repository "

+ event.getRepoId());

}

5. Add copy of the com*.kintosoft.svnwebclient.jira.public\_interface.CommitsEventbusService* interface into your project. This interface allows to (un) subscribe (from) to the Eventbus provided by Subversion ALM. There is an example available in the com.kintosoft.jira.svn.MyPluginComponentImpl class. The *CommitsEventbusService* can be injected in the constructor:

public void register() {

eventBus.startListenting(commitListener);

log.info("MyPluginComponentImpl commit listener registered");

}

@PreDestroy

public void unregister() {

log.info("MyPluginComponentImpl destroying");

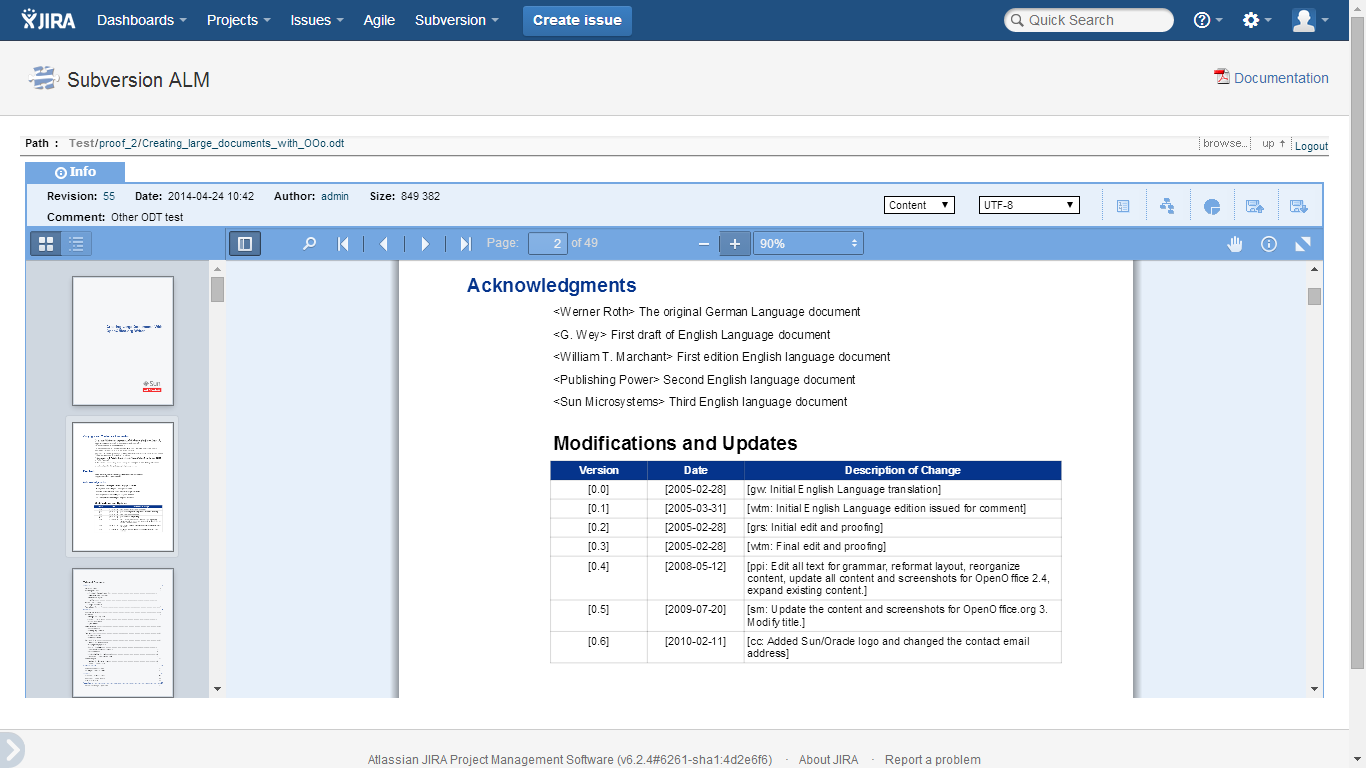
eventBus.stopListenting(commitListener);

log.info("MyPluginComponentImpl commit listener unregistered");

}

# Office and PDF documents

Since the 6.0 version, Microsoft Office and OpenOffice documents are supported. In the 6.0 version they will be displayed by using an internal viewer. In further versions new features like version comparison will be supported:



Documents are transformed into PDF files and stored in the plug-in cache *kintosoft\_subversion\_doc* under the JIRA HOME directory for performance.