Save & Restore

Design Proposal

|  |  |
| --- | --- |
| Revision: | 0.1 |
| Status: | Released |
| Project: |  |
| Document ID: | CSL-DOC-15-123998 |
| File: | SaveAndRestore.docx |
| Owner: | Jaka Bobnar |
| Last modification: | February 26, 2016 |
| Created: | October 9, 2015 |

Document History



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Changed/reviewed | Section(s) | Modification |
| 0.1 | 2015-10-12 | jbobnar | All | Document created |
| 1.0 | 2016-02-23 | jbobnar | All | Updated according to the latest implementation |

Confidentiality

This document is classified as a public document. As such, it or parts thereof are openly accessible to anyone listed in the Audience section, either in electronic or in any other form.

Scope

This document describes the design proposal for save and restore application at FRIB.

Audience

This document is targeted to FRIB and Cosylab engineers.

Table of Contents

[1. Introduction 4](#_Toc444253349)

[2. Use-cases & Requirements 5](#_Toc444253350)

[2.1. Use-cases 5](#_Toc444253351)

[2.2. Requirements 5](#_Toc444253352)

[3. Design 6](#_Toc444253353)

[3.1. Data Storage 6](#_Toc444253354)

[Snapshots 6](#_Toc444253355)

[Beamline Sets 6](#_Toc444253356)

[Base Levels 6](#_Toc444253357)

[Branches 6](#_Toc444253358)

[3.1.1 Repository Structure 7](#_Toc444253359)

[3.1.2 Data Format & File Contents 8](#_Toc444253360)

[Beamline Sets 8](#_Toc444253361)

[Snapshots 9](#_Toc444253362)

[Tags 10](#_Toc444253363)

[3.1.3 User Interface 10](#_Toc444253364)

[3.1.4 Code Structure 11](#_Toc444253365)

[Plugins and Features 11](#_Toc444253366)

[Extension Points 12](#_Toc444253367)

Glossary of Terms

|  |  |
| --- | --- |
| EPICS | Experimental Physics and Industrial Control System |
| PV | Process Variable |
| CS-Studio (CSS) | Control System Studio |
| Beamline set | Configuration that defines a set of PVs |
| Snapshot | A recording of PV values at a specific time. The set of PVs is defined by the beamline set. |

References

1. Git: <https://git-scm.com/>
2. EPICS: <http://www.aps.anl.gov/epics/>
3. Control System Studio: <http://controlsystemstudio.org/>
4. E(fx)clipse: <http://www.eclipse.org/efxclipse/index.html>
5. CSV File specifications: <https://tools.ietf.org/html/rfc4180>

# Introduction

Save and restore is an advanced control system operator tool, which can be used during all stages of commissioning and operation of the machine or its parts. Its primary purpose is to record and store the values of different control system endpoints in order to examine, compare or restore them at any later time.

The main users of the application are machine operators and machine physicists, who need a comfortable and easy mechanism to bring the machine into a known state. User can browse the sets of existing machine snapshots and apply the values from any snapshot to the control system endpoints. At the same time user is allowed to create a new snapshot of the current machine configuration in order to be able to reuse it later. Special tagging of the snapshots (e.g. *Golden Snapshot*) allows classifying the snapshots according to certain criteria.

In addition to saving and restoring, the GUI also provides means to compare the values of different snapshots and adjust the values before restoring them.

# Use-cases & Requirements

## Use-cases

The following use-cases have been provided by users:

1. User is able to select beam types (branches) to save PV sets
2. User is able to compare current PV setting to those previously saved
3. User is able to compare previously saved PV sets
4. User is able to scale a PV sets before applying sets
5. User is able to select or deselect magnet types for scaling and applying sets (channelfinder)
6. User is able to make a log entry referencing save set with comments
7. User is able to compare local changes to PVs to reference set
8. User is able to compare local changes to PVs to design optics
9. User is able to make local changes and undo changes to previous commit
10. User is able to save beam line sets, and save configuration of beam lines
11. User is able to select pvs context to send to configuration
12. User can add "read only" pv for comparison
13. User can tag save sets (great save, lattice configuration)
14. API needs to allow changing of save set data column (load model, python scaling, etc.)

## Requirements

From the above use-cases we derive the following requirements:

1. It shall be possible to define and save a beamline set (configuration).
2. It shall be possible to take a snapshot of PV values for a selected beamline set and store it.
3. It shall be possible to set the values from a snapshot to corresponding PVs.
4. It shall be possible to edit the values in the snapshot before saving or before setting the values to PVs.
5. It shall be possible to modify (e.g. scale) the entire set of values in a snapshot through a public API.
6. It shall be possible to exclude PVs from actions (e.g. excluded PV shall not be modified (i.e. scaled), PV value shall also not be set when the snapshot is restored).
7. It shall be possible to compare values of any two snapshots and design optics.
8. It shall be possible to add a read-only PV to the snapshot PV list and display the archived value of the PV at the time when the snapshot was taken.
9. It shall be possible to tag the saved snapshots.

# Design

Save and Restore will be implemented as a two tier application and will access other services in order to retrieve information required to perform the necessary tasks. The primary part of the application is the storage facility, which will contain all shared information, such as the beamline sets, snapshots, tags and comments etc. The second part of the application is the user interface, which will be connected to the storage and will have read and write access for it.

Git [1] will be used in initial implementation of the data storage, while the GUI will be implemented as part of the Control System Studio [3].

## Data Storage

The data storage part of save and restore will be implemented in Git. Git is a distributed revision control system, primarily used for managing source code. It offers a convenient way to track changes to files, compare different versions, searching, branching etc. These features will be wrapped by a client library, which will hide the git logic and offer them as required by the GUI.

A central repository will be created, which will hold all beamline sets and snapshots ever created and to which all clients will have access. Client Application will create a local clone of this repository and keep it updated with the central repository. This will give the client access to all files. User will be able to make changes to the files, such as editing a beamline set or taking a new snapshot of PV values for a particular beamline set. When satisfied with the result, user will provide a commit log and commit the changes to the repository. All changes will then be pushed back to the central repository and made available to other clients. A single commit will always contain only one file.

#### Snapshots

Every snapshot that is taken by the client application will be included in its own commit with a commit comment and unique hash (generated automatically by git). This will allow to track the changes to the snapshots and to retrieve any version of the snapshot that was ever made. The commit message should be elaborative enough to explain why the snapshot was taken.

#### Beamline Sets

Similarly to snapshots, user will be allowed to modify and save beamline set file. Each modification could either be stored as a change to the existing beamline set or as a new file. When the modification is stored as a change, the beamline set will keep its reference to the snapshots, even if a snapshot now contains different PVs. If the beamline set file is stored as a new file, no snapshots will be associated with it.

#### Base Levels

Base levels are the top most directories in the repository. They are used to separate the sets and snapshots that belong to different experiments or setups.

#### Branches

Branches can be used to separate particular beamline sets and snapshots from the common user. In day-to-day operation all users will work on the same git branch. However, at certain times (e.g. testing of a new equipment) users might want to create beamline set files and snapshots, which they do not want them to be visible to the operators. In such case, user will be able to switch to a new git branch. All changes made on that branch will only be visible there, unless explicitly merged into the master branch. The same approach can also be used during major machine upgrades; using a new branch in such case will clearly distinguish which snapshots belong to the pre-upgrade and which to post-upgrade era.

|  |  |
| --- | --- |
|  | Initially it was suggested that branches could be used to separate snapshots that belong to different experiments or elements. On the positive side this would clearly separate which versions of snapshots and beamline sets belong to a particular experiment. On the other hand, it would present a significant overhead when trying to load and compare snapshots from different experiments. Loading the file contents from git repository requires first to check-out the entire branch, which (depending on the structure of the branches) may require several seconds to complete.  In addition, we use git and file system for practical reasons, so that anyone can easily access any file. While branches are not a rocket science, working with them can be dreadful, for someone who is not familiar with git. |

### Repository Structure

Most of the actions performed on the git repository will be done by the Save and Restore GUI client and will be hidden from the user. However, some expert users might need to manage the files manually; therefore, the structure of the file system is important. Two different types of files will exist in the system: beamline sets files and snapshot files. In order to distinguish them they will have different file extensions:

* Beamline set file will be described by the extension .bms (BeaMline Set)
* Snapshot file will be described by the extension .snp (SNaPshot)

The most common search operation on the repository will be to locate the list of beamline set files that match certain criteria (e.g. all sets valid for specific isotope of Pb, with a specific charge). While git offers search mechanisms that can be useful in such cases, the structure will be kept user friendly – the beamline sets files will be located in their own directory and so will be snapshot files. To further organize the files, they can (optionally) be placed inside subdirectories that more precisely specify what part of the system they relate to.

Snapshots will be placed in a mirrored subdirectories structure located within the snapshot directory. The file name of the snapshot will always be identical to the filename of the beamline set file.

Below is an example of a repository file structure:

* Ag\_105\_10p
  + BeamlineSets
    - <Set 1>.bms
    - <Set 2>.bms
  + Snapshots
    - <Set 1>.snp
    - <Set 2>.snp
* Sn\_117\_16p
  + BeamlineSets
    - <Set 1>.bms
    - <Set 2>.bms
  + Snapshots
    - <Set 1>.snp
    - <Set 2>.snp

The names of the files are not prescribed, but should be such that the file can be easily identified; the name should give a hint about the file content. In the top most levels the name is constructed from the element name, the mass of the isotope and the charge of element (followed by p for positive or n for negative charge). The top most level names proposal is only a recommendation, but it is up to the users to name the levels. However, if arbitrary naming convention is used, some UI browsers (e.g. periodic table) might not be able to retrieve the names.

### Data Format & File Contents

#### Beamline Sets

Beamline set files specify the list of PVs that will be stored by Save & Restore. They are essentially a list of entries, where each entry could have additional predefined set of properties (the same for all entries) that define how that particular entry it is treated (e.g. a PV is marked to be read only). Though there are currently no such requirements, the proposed model allows such extensions.

Each beamline set file contains a comment, which explains what the file is about, why was it created or any other information that could be valuable to the user. In line comments are also possible (denoted by ‘#’ character at the beginning of the line) in the sense of not obstructing the Save & Restore application, but they will not be displayed or preserved by the application.

To keep the contents of the file simple and as human-readable as possible, a comma separated value format will be used to store the beamline set file. The following example demonstrates how the beamline set file will look like:

# Save & Restore

#

# Description:

# A template for a beamline set file to demonstrate how the file should

# look like.

#

PV,READBACK,DELTA[,OTHER POTENTIAL PROPERTIES]

PV:SET\_1,PV:READ\_1,10

PV\_SOMETHING,,

# This is an in-line comment which will be ignored

PV\_FOO,PV\_BAR,

Currently, three columns are defined for the set files:

* PV: the name of the primary PV,
* READBACK: the name of the readback PV,
* DELTA: the threshold value, which defines when the values for this entry are close enough to be treated as equal.

The only obligatory column is PV, while READBACK and DELTA are optional. However, if DELTA is defined, the READBACK has to be defined as well. The value of READBACK and DELTA can be empty for any entry, which means that they are not defined for that entry.

If any of the entries in the file contains the comma (‘,’) character, the field that contains it should be wrapped in quotes as it is in standard CSV format [5]. Line breaks are not allowed!

#### Snapshots

Similar to the beamline set files the snapshots are also lists of entries where each entry has a set of properties, such as PV name, value, timestamp, status, severity etc. Snapshots will be created, examined, and compared in spread sheet or grid views; therefore a spread sheet file format seems a natural choice. In order to keep the file easily “accessible” by other clients and tools, they will also be stored in CSV format.

In EPICS V3 the PV value can be one of the following: integer, floating point number, character, string, or an array (waveform) of any of those. Any of these values can be easily converted into a text format. Waveforms can be stored as [e1; e2; e3; …; en], where ei is the i-th element of the waveform arrays [2]. To avoid confusion a semi-colon (rather than a comma) is used for separation of waveform elements, because comma is used as a general column separator.

In addition to the actual PV value, each EPICS event also provides the timestamp (since epoch) of the value, severity, and status. All these will be stored when a snapshot is taken.

EPICS V4 adds additional composite data types, where a value of a PV can be a combination of different base types. JSON formatting provided by DIIRT can be used for such data types.

Below is an example of a snapshot file.

# Date: 2015-08-13T13:30:22

PV,TIMESTAMP,STATUS,SEVERITY,VALUE\_TYPE,VALUE,READBACK,READBACK\_VALUE,DELTA

PV\_SET,1449840706.555344346,LOW,MINOR,double,5.3,,,

PV\_SET1, 1449840707.555344346,OK,OK,double array,”[2.8;1.2;7.3;2.1]”,,,

PV\_STR, 1449840708.555344346,OK,OK,string,”some string value”,,,

The following columns are defined for a snapshot file:

* PV: the name of the PV
* TIMESTAMP: the timestamp of the stored PV value
* STATUS: status of the stored PV value
* SEVERITY: alarm severity of the stored value
* VALUE\_TYPE: the type of value (double, long, string, double\_array etc.)
* VALUE: the stored PV value
* READBACK: the name of the associated readback PV (as defined in the set file)
* READBACK\_VALUE: the value of the readback PV at the time when the snapshot was taken
* DELTA: the threshold value used for comparison of values related this entry

PV, READBACK, and DELTA values are repeated from the set file. Theoretically they could be omitted; however, the set file can change, but we would still like to use the snapshots that belong to the previous set file. While all that information is stored in git database, it can be extremely cumbersome to retrieve it. In addition to that, if the data are repeated in the snapshot, the file is self-sufficient and therefore fully transferrable.

|  |  |
| --- | --- |
|  | Initial proposal stated that the entire snapshot can be stored as a VTable. However, the VTable is limited to contain single-type values per column, which means that a beamline set could only contain double type PVs, or only string type PVs etc. In addition, the output format of a VTable is not the most human-friendly. Currently there is no data type in CSS that could hold a matrix of mixed columns. |

When different values of the same PV are compared between each other there are different criteria when two values are equal. For example, digital signals have to have identical values while analog are still identical, when the difference is within certain limits. The DELTA values in the beamline set specify what threshold values are used. If the DELTA is 0.1, the values are considered identical when ABS(Value1 - Value2) < 0.1.

Sometimes the threshold value might also depend on the value itself. For example, when the value is small, the threshold is also small, but when the value is higher, the threshold can be higher as well. Therefore, DELTA will also support function values.

#### Tags

Tagging of snapshots can be implemented by employing the git tagging mechanism. A tag in git is essentially naming of a specific commit. In save and restore case each commit will contain a single file. When tagging that commit with a specific name, the tag can easily be associated with one single file.

Tags in git have to have unique names. Therefore it is necessary that the tag name contains the entire path to the file which. The tag of the file test/example.snp for element Pb would be: (master)Pb/test/example(<tag\_name>), where <tag\_name> is the name given by the user. The tag message can contain further information about this tag.

### User Interface

User interface will be done as a standalone perspective in control system studio. It will be composed of at least 3 different views or editors:

* Browser: allows browsing for available snapshots or beamline sets.
* Snapshot viewer: shows the content of the snapshot file, provides interface to take snapshots, restore values, compare values etc.
* Beamline set files editor: provides a mechanism to create new or edit existing beamline sets.

GUI will use an abstraction layer to access the data from the git storage. This will later allow replacing the storage with other services if necessary (e.g. MASAR). Other extension points will also be provided, through which different data or different user interface parts could be provided.

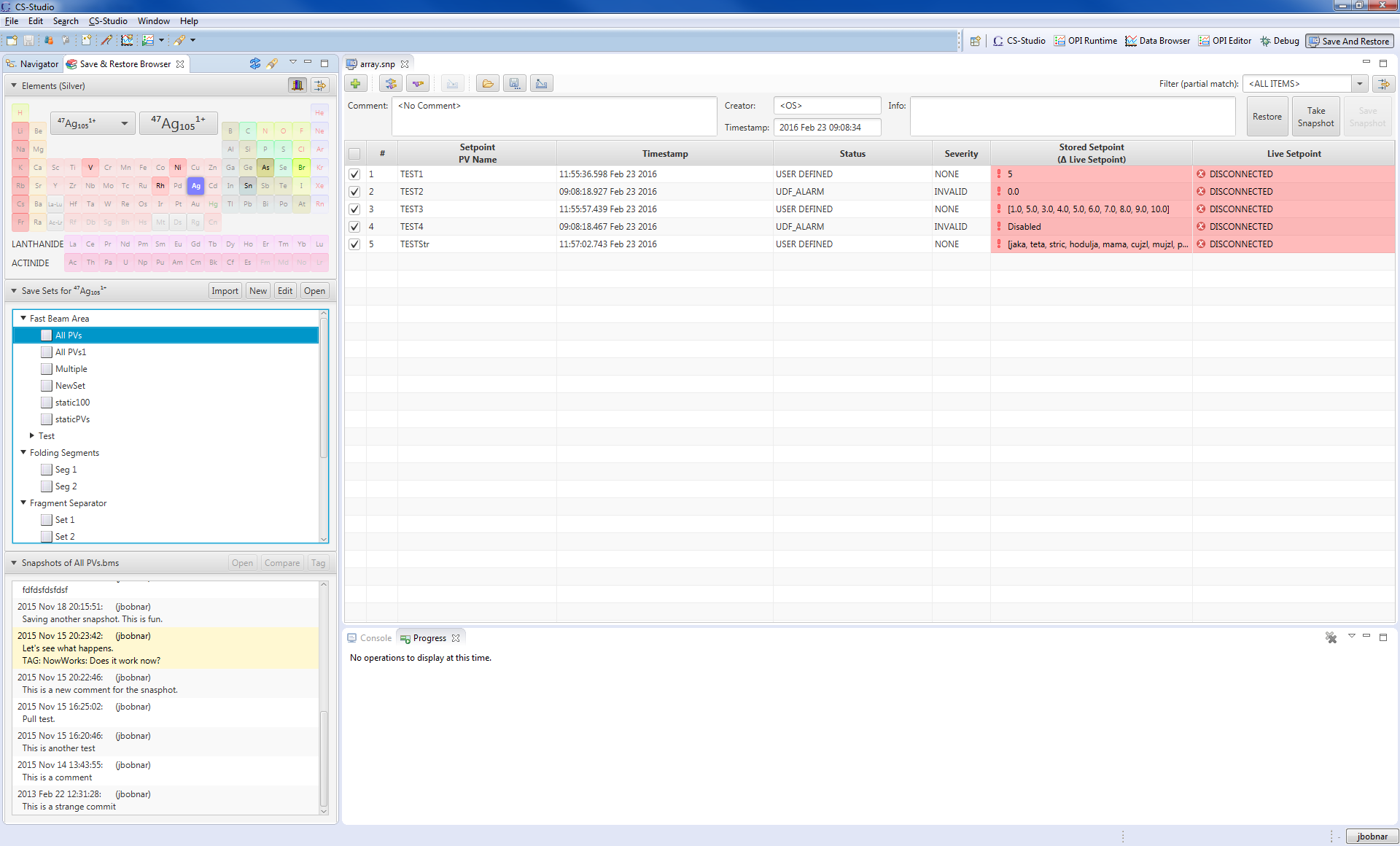


Figure 1: The browser view and the snapshot viewer

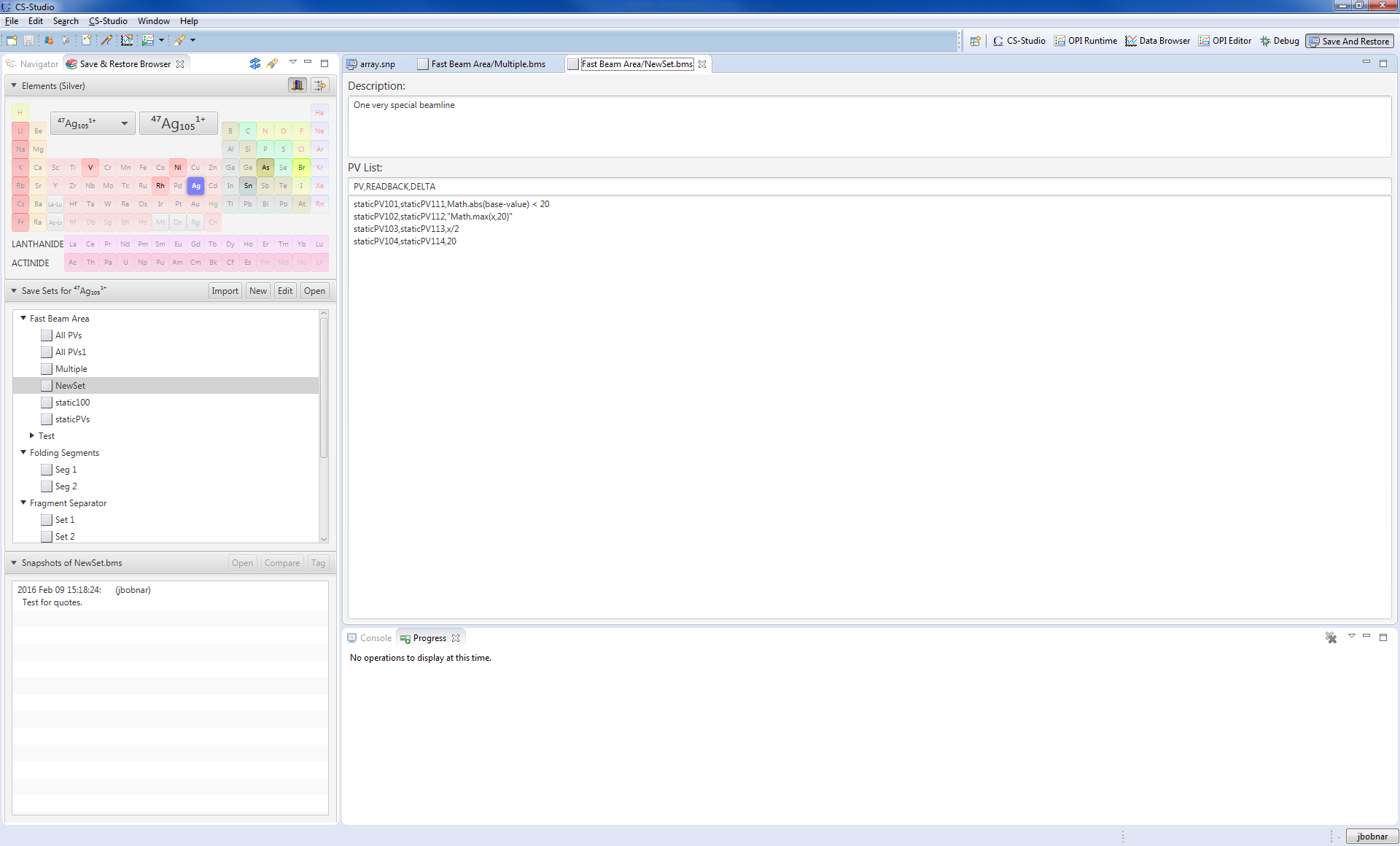


Figure 2: The browser view and beamline set editor

### Code Structure

#### Plugins and Features

The following plugins will be created (if not stated otherwise, each plugin may depend on one or plugins listed in the list above it):

* org.csstudio.ui.fx.util: provides generic JavaFX ui components, such as message dialogs, utility classes, charts.
* org.csstudio.saverestore: provides the general data structures, defines the data provider interface and extension point and implements general help classes and methods to work with data (parsing, exporting, executors)
* org.csstudio.saverestore.git: implementation of the ..saverestore.dataprovider extension point that uses git as the underlying storage system. No other plugin depends on this plugin.
* org.csstudio.saverestore.masar: implementation of the ..saverestore.dataprovider extension point that uses MASAR as the underlying storage system. No other plugin depends on this plugin
* org.csstudio.saverestore.ui: provides the snapshot viewer and the beamline set editor.
* org.csstudio.saverestore.ui.browser: provides the browser view for browsing and selection branches, base levels, beamline sets and snapshots.
* org.csstudio.saverestore.ui.browser.periodictable: provides implementation of the ..ui.browser.baselevelbrowser extension point as a periodic table.
* org.csstudio.saverestore.help: provides the eclipse help pages for the entire application

Several test fragments also exist, which implement junit tests for their host plugins.

The plugins are grouped into features, which can be installed independently, providing that all dependencies are resolved:

* org.csstudio.saverestore.feature: includes the basic ui and non ui plugins without any data provider. The feature depends on the ..archive.diirt.datasource and ..logbook feature
* org.csstudio.saverestore.git: includes the git data provider. It depends on the org.eclipse.jgit and org.csstudio.saverestore.feature.
* org.csstudio.saverestore.masar: includes the masar data provider plugin. It depends on the org.csstudio.saverestore.feature, but it also requires the plugins org.epics.pvaccess and org.epics.pvdata version 4.0.0 or newer.
* org.csstudio.saverestore.periodictable.feature: includes the periodic table plugin. It depends on the org.csstudio.saverestore.feature.

#### Extension Points

The following extension points are defined in the saverestore application. Refer to the extension points schema definitions to obtain more details. Refer to javadoc to obtain more details about the interfaces related to each of the extension points.

* org.csstudio.saverestore.dataprovider provides a mechanism to plug in different sources for the save sets and snapshots, such as git, or MASAR. The provider is identified by a unique ID, a readable name and description and a class, which implements the org.csstudio.saverestore.DataProvider interface.
* org.csstudio.saverestore.ui.baselevelvalidator provides a mechanism to validate the name of all created base levels (element names). Whenever a user enters a new storage name the validator is used to verify if the name conforms to the standards.
* org.csstudio.saverestore.ui.parametersprovider allows to plug-in a mechanism to retrieve the readback names and pv threshold values. The extension provides an implementation of the org.csstudio.saverestore.ui.ParametersProvider interface.
* org.csstudio.saverestore.ui.valueimporter provides a mechanism to import values from third party sources, such as optics data. The data imported via this mechanism are added to the open snapshot view. The extension point needs to provide an implementation of org.csstudio.saverestore.ui.ValueImporter interface. There can be more than one such implementation in the product.
* org.csstudio.saverestore.ui.browser.baselevelbrowser provides an additional implementation of the base level browser UI. This can be any kind of (JavaFX based) component that allows browsing and selecting the base levels, such as for example the periodic table.