

# Architecture of BVD

- Purpose of this Document
- Server-side
  - Models
    - src/pull/models.py
  - Views - Helper Functions and Data Processing
    - src/pull/views.py
    - src/jenkins/jenkins.py
- Client-side
  - The Page
  - Widget Rendering
    - src/pull/static/js/widget.js - (Widget.\*)
    - src/pull/static/js/render.js - (Widget.render.\*)
    - src/pull/static/js/utills.js - (BVD.utills.\*)
  - Window Logic
    - src/pull/static/js/modals.js
    - src/pull/static/js/jobs.js
    - src/pull/static/js/edit\_readonly\_display.js
  - Other Utilities
    - src/pull/static/js/bvd.js
    - src/pull/static/js/utills.js
    - src/pull/static/js/data.js
  - Javascript/jQuery Libraries
- Communications
- Useful Links
  - Rally
  - SSL
  - Freetile (used to arrange widgets and products)

## Purpose of this Document

This document serves as documentation of the architecture of BVD so that future developers have a reference for the structure of the code aside from the code itself. It contains three subjects: a description server-side code which executes entirely on the server, a description of client-side code, and a chart of the client-server exchange of data over HTTP.

## Server-side

### Models

#### src/pull/models.py

- CiServer - Represents a known Jenkins server
  - hostname - the hostname of the Jenkins server
- UserCiJob - Represents an individual Jenkins job/widget
  - user - foreign key for the user which owns the widget
  - ci\_server - foreign key for the CiServer the job is located on
  - jobname - the name of the job on the Jenkins server
  - status - the status of the job
    - **Deprecated - this is now stored in memory cache**
  - displayname - the name of the widget that's displayed on BVD
  - icon - the image that is displayed when the job's status is 'SUCCESS'
  - readonly - the readonly status of the widget
    - **Deprecated - there's really no reason for this to be in the database**
  - entity\_active - the 'active' state of the widget.
  - appletv - whether the widget is displayed to readonly users
  - appletv\_active - whether the widget is active when displayed to readonly users
    - used to hide the widget to readonly users but not remove it from the list of widgets in the Edit Public TV window
- Product - Represents a group of widgets
  - productname - the name of the product
  - jobs - a many-to-many field to store the list of jobs associated with the product

- show\_rally - whether to show a rally chart for this product
- rally\_release\_name - a substring of the Rally Release which can be displayed for the product

Each model has a corresponding Form found in **src/pull/forms.py**. Forms are used to load and save rows in the database/instances of the model and to validate user input. [Additional documentation on Forms can be found here.](#)

## Views - Helper Functions and Data Processing

Functions which return an HTTP Response are not listed here. They can be found in the Communications section.

### src/pull/views.py

Function Name	Arguments	Purpose
append_http	hostname	Should really be named "prepend_http" or "append_to_http" because it prepends the http protocol to the hostname argument if it does not exist and returns the result.
widget_to_dictionary	widget	Returns a Python dict() of a job/widget instance. Should really be used anywhere jobs are loaded to be sent to the client, but in practice is only used in pull_jobs_for_product() and save_widget(). Consider refactoring to simplify the structure of BVD's data processing code.
get_jobs_for_readonly	only_active=True	Returns a data structure of products and jobs viewable by readonly users/Apple TV/the public BVD displays. Also used to load the widget list for the 'Edit Public TV Display' window, hence the argument which allows non-active widgets to be loaded as well.
get_jobs_for_user	user	Returns a data structure of products and jobs belonging to the user.
get_rally_release_oid	release_name, workspace, username, password	Returns the Rally ObjectID of the most recent 'active' release with the substring release_name with the workspace and credentials given.

### src/jenkins/jenkins.py

These functions are methods of the **RetrieveJob** class. The hostname and jobname are given to the constructor of a **RetrieveJob** object. There isn't really a good reason for it to be structured like this as far as I can tell.

Function Name	Arguments	Purpose
_parse_jenkins_timestamp	timestamp (and optional auth header values)	Parses the timestamp returned from Jenkins' JSON api and returns a python datetime object
_get_time_diff	time	Returns a human-readable difference in time between the given timestamp and the present time
lookup_hostname	optional auth header values	Determines if the given hostname is accessible by BVD

lookup_job	optional auth header values	Determines if the given job can be found and returns the status of the job
lookup_last_successful_build	optional auth header values	Looks up the last successful build of the given job and returns data on the timestamp

## Client-side

Though BVD is ostensibly a "Python-on-Django" web app, the majority of the project is implemented in Javascript because the client-side interface is really the entire point of BVD and where most of the work to make the actual display is done. As a result, and perhaps partially because of the fact that HTML, Javascript, and the DOM weren't really designed for the task of dynamic, responsive user applications, it's somewhat hard to untangle the pure client-side code with the code that communicates with the server and to categorize the resulting code further.

I've done my best to use a number of architectural ideas in BVD's client-side code:

1. The whole point of the server is to manage data, so don't try to do anything the server's already done. Therefore, instead of updating the client-side data, throw it out and load it from the server again.
2. Load data through AJAX and JSON and create DOM objects based on that instead of templating. This way you at least have a reference to the object in javascript when you create it instead of trying to hunt for it through jQuery selectors.
3. Where possible, use an HTML <form> to send data to the server. [Django's Forms](#) make this really easy.

There are a couple places where these rules were broken, either in code I inherited and never was able to update or where it simplified the code to break the rule. The latter case specifically applies to the Edit Public TV window where the user's widgets are loaded via template, half-breaking rule 2 (because the template is loaded via AJAX).

## The Page

When the page is loaded, BVD initializes itself by creating a timer-based callback to keep the page updated. This is done with inline javascript in the `index.html` template and the `Poll` class in `src/pull/static/js/bvd.js`

## Widget Rendering

This section describes the client-side code which handles the rendering of widgets on the screen.

### `src/pull/static/js/widget.js` - (Widget.\*)

Function	Arguments	Purpose
init	hostname, jobname, displayname, status, id, counter, readonly, background_img, timeSinceLastSuccess, product	Constructor, creates the widget and menu and sends it to <code>Widget.render.add_widget()</code>
make_icon		Internal function called by constructor, builds the menu for the widget.
set_status	status	Internal function, calls one of <code>make_success</code> , <code>make_error</code> , <code>make_unstable</code> , or <code>make_down</code> to set the widget status

New widget objects are created by `BVD.utils.draw_widgets()`

### `src/pull/static/js/render.js` - (Widget.render.\*)

These functions are used for rendering and manipulation of individual widgets on the screen.

Function	Arguments	Purpose
----------	-----------	---------

add_widget	widget, product, readonly	Displays the widget, called by the Widget constructor. If product is undefined it renders the widget directly to the page, otherwise adds it to a product.
add_product	productname, readonly	Called by add_widget if the product <div> doesn't exist yet. Readonly determines whether the product's menu is displayed or not.
refresh_grid	animate	Sets the size of the widgets with BVD.utils.set_size_of_widgets() and calls the freetile jQuery library to lay out the widgets.
remove_widget	widget	Removes the widget from the display
resize	widget, width, height	Resizes the widget. Called by BVD.utils.set_size_of_widgets()

## src/pull/static/js/utlis.js - (BVD.utlis.\*)

These functions are used for rendering and manipulation of all the widgets on the screen.

Function	Arguments	Purpose
remove_old_widgets		Removes all the widgets from the screen. Used by Poll.ajax() to refresh the screen.
set_size_of_widgets	count	Determines the optimum size of widgets based on the number of widgets.
draw_widgets	data	Loops over the data and creates the widgets.

## Window Logic

This section describes the client-side code which handles the rendering and UI logic of the popup windows ("modals") used to interact with BVD.

### src/pull/static/js/modals.js

Function	Arguments	Purpose
BVD.modal_factory	url, id, opts	Used to create the windows (modals) by rendering the template with jQuery UI.

The rest of this file creates the buttons and button callbacks for **index.html**

### src/pull/static/js/jobs.js

Function	Arguments	Purpose
BVD.jobs.add_job	txt_map	The function to create the Add Job modal.

There isn't really a reason for this to be in its own file. Consider refactoring.

### src/pull/static/js/edit\_readonly\_display.js

This file contains some of the logic for the Edit Public TV window. The prefix naming conventions for these is: (erd -> edit readonly display), (erdd -> edit readonly display, public display widgets), and (erdu -> edit readonly display, user widgets).

Function	Arguments	Purpose
erd_display_line	pk, displayname, appletv_active, productname	Renders a list element to the list of public display widgets.
erd_display_product	productname, erd_line	Creates a <div> for a new product to group widgets in the public display together.
change_erdd_line_active	pk, appletv_active	Toggles the transparent style of list elements in the list of public display widgets.
update_erdd		Updates the list of public display widgets in the Edit Public TV window.

## Other Utilities

This section describes any other code which can't be categorized into the other two groups.

### src/pull/static/js/bvd.js

Function	Arguments	Purpose
Poll.ajax	url	Pulls the list of widgets from the server and uses it to refresh the display.

### src/pull/static/js/utlis.js

Function	Arguments	Purpose
BVD.utlis.do_ajax	type, url, data, success, error	Provides a simplified interface for jQuery's \$.ajax() function.

### src/pull/static/js/data.js

Function	Arguments	Purpose
BVD.data.get_url	url, querystring	Gets a url for an action. Probably can be factored out.

## Javascript/jQuery Libraries

Library	Purpose
jquery.fastLiveFilter.js	Used to filter jobs in the Add and Edit Product windows.
jquery.freetile.js	Used to lay out products and widgets into a grid.
jquery.slides.js	Used to slide between the job status page and the Rally ReleaseCumulativeFlowDiagram page.

## Communications

Server-side response functions, unless otherwise prefixed, exist within the namespace `bvd.pull.views.*`

Client-Side Request Source	Method	Server-Side Response Function	Data Sent	Data Received	Purpose
poll.ajax('/pull/pull_jobs/')	GET	pull_jobs		a data structure of jobs	To pull the list of jobs for a user
poll.ajax('/pull/pull_apple_tv_jobs/')	GET	pull_apple_tv_jobs		a data structure of jobs	To pull the list of jobs for readonly users
BVD.modal_factory()	GET	get_modal	template (the name of a template)	html (the content of the "modal")	To load the content of a window/modal
update_erdd()	GET	pull_all_display_jobs		a data structure of jobs	Loads the list of public display jobs for the Edit Public TV window.
view_product.html - update()	GET	pull_jobs_for_product		a data structure of jobs	Loads the list of jobs for a specific product.
add_job.html, edit_job.html	POST	validate_hostname	hostname, username	HTTP status	Validates that a Jenkins hostname is valid
login.html	POST	validate_username	username	HTTP status	Validates that a BVD username is valid.
add_job.html, edit_job.html	POST	validate_job	hostname, jobname	HTTP status	Validates that a Jenkins job and hostname is valid.
add_job.html	POST	add_job	<form> data	HTTPResponseRedirect('/')	Creates a new job.
'Remove Widget' menu option	POST	remove_job	job id (primary key)	HTTP status	Deletes a job.
	POST	autocomplete_hostname	partial hostname	list of possible hostnames	<b>Deprecated</b> no longer used by Add Product window
login.html	POST	login	username, password	HTTP status	Logs in to BVD.
Logout button	POST	logout		HTTP status	Logs out of BVD.
edit_job.html,	POST	save_widget	<form> data	HTTPResponseRedirect('/')	Edits an existing job.
add_product.html	POST	add_product	<form> data	HTTPResponseRedirect('/')	Adds a new product.
edit_product.html	POST	save_product	<form> data	HTTPResponseRedirect('/')	Edits an existing product.
'Remove Product' menu option	POST	remove_product	productname	HTTPResponseRedirect('/')	Deletes a product.
'Edit Image' menu option	POST	edit_widget_image	widget_id, image file	pull_jobs()	Uploads an image for the job.

/appletv=1, view_rally.html	GET	pull_rally_for_applet v		a data structure of Rally Release ObjectIDs	Pulls the list of Rally Release ObjectIDs for publicly-accessible products.
view_rally.html	GET	pull_rally		a data structure of Rally Release ObjectIDs	Pulls the list of Rally Release ObjectIDs for a user's products.

## Useful Links

Future developers of BVD might find these links useful:

## Rally

- <https://help.rallydev.com/loginkey>
- <https://help.rallydev.com/apps-outside-rally>
- <https://rally1.rallydev.com/slm/doc/webservice/objectModel.sp>

## SSL

- <http://stackoverflow.com/questions/8023126/how-can-i-test-https-connections-with-django-as-easily-as-i-can-non-https-connec>
- [http://www.akadia.com/services/ssh\\_test\\_certificate.html](http://www.akadia.com/services/ssh_test_certificate.html)

## Freetile (used to arrange widgets and products)

- <http://yconst.com/web/freetile/>