# AllegroGraph Web View — Reference Guide

The AllegroGraph Web View is an interface for exploring, querying, and managing AllegroGraph triple stores through a web browser.

## Basic usage

The web view can be loaded in two ways -- either simply `load` the file `load.lisp` that comes with the source code, or set up your ASDF to be able to find the `db.agraph.webview.asd` file (as well as the `.asd` files for the st-json and salza2 libraries), and load it with `(asdf:oos 'asdf:load-op :db.agraph.webview)`.

If you already have a triple store open, and the variable `db.agraph:*db*` points to it, the web server can be started with:

```
(db.agraph.webview:start-webview (make-instance
'db.agraph.webview:simple-site :db db.agraph:*db*))
```

This will try to start the server on port 80. If you do not have access to that port, pass `start-webview` a `:port` keyword argument to specify another port.

Now direct your web browser to http://localhost/ (or something like localhost:8080 if you specified another port). You will see some general information about the store, and can try some SPARQL or Prolog queries by clicking 'Queries: new' in the navigation bar. See below for a more detailed description of the interface.

## Programming interface

The AllegroGraph Web View runs in an AllegroServe server. The most straightforward way to start a server is...

```
start-webview (site &key hostname port)
  → server
```

Starts a server and publishes the given site in it. `port` defaults to 80. See below for more information about site objects.

```
publish-webview (site server)
```

Often you'll need to have more control about the way the server is created (as per net.aserve:start), or you'll want to publish several applications in the same server. This function publishes a site in a given server. See the :prefix argument to a site for a way to prevent URLs from clashing.

Site objects hold the configuration and state used to run a web view. There are three types of sites:

```
make-instance 'simple-site (&key db spec name blurb
  namespaces notable queries use-reasoner prefix allow-
  prolog google-key)
  → site
```

Creates a simple 'appliance mode' site that is configured entirely programmatically. The arguments are mostly optional, and have the following meaning:

`db`
: The triple store to serve. Should be an instance of db.agraph:abstract-triple-store. Either this or `spec` should be given when creating a simple site.

`spec`
: Use this to have the web view open the triple-store itself (and close it again when closed). Can have the form `(:local filename write-p)`, which opens a local file, in read-only mode if `write-p` is `nil`, or `(:remote filename host port write-p)`, which opens a remote store through a Lisp socket connection.

`name`
: Give the store a name, which overrides the filename as the name shown in the interface.

`blurb`
: A short text describing the database. Shown in the overview.

`namespaces`
: A list of `(abbreviation url)` lists determining the namespaces used by the web view. These will cause matching URLs to be abbreviated (as in `rdf:type`), and can be used when entering queries.

`notable`
: A list of 'notable' nodes, in `(node description)` form. Will be shown in the store overview, and can be used to give people a starting point to explore the store. The nodes should use either be AllegroGraph parts (!-notation), or strings using an N-triples-like notation. Descriptions can be arbitrary strings.

`queries`
: A list of `(type name query)` lists, where `type` is either `"SPARQL"` or `"Prolog"`, and `name` and `query` are arbitrary strings. These will be shown as 'pre-defined queries' in the store overview.

`use-reasoner`

Configures whether reasoning is available when querying the store. Set to `t` to use the default `rdfs reasoner`, or to a symbol to use a specific reasoner. When `nil` (the default), no reasoning is used.

`prefix`
: Sets a URL prefix to publish the web view under. This should start and end in a slash. Defaults to `"/"`.

`allow-prolog`
: Enables Prolog queries on this site. Defaults to `nil`, since reading and executing Prolog is (theoretically) not entirely secure.

`google-key`
: Sets a key for use with the Google Maps API (create one here) for displaying geospatial data. When left at `nil`, the map functionality is disabled.

```
make-instance 'site (&key db spec cache-file prefix
  allow-prolog google-key allow-new-accounts-p
  anonymous-users-p)
  → site
```

Sites of class `site` use AllegroCache to store persistent data. Such sites allow user accounts and can be configured through the web interface. See `add-admin` for a way to give yourself a superuser account. The `db`, `spec`, `prefix`, and `google-key` arguments work as in `simple-site`.

`cache-file`
: A string that determines where on the filesystem the AllegroCache data is stored. If this directory does not exist, it is created as the site is initialized.

`allow-new-accounts-p`
: Configure whether users are allowed to create their own accounts.

`anonymous-users-p`
: Determines whether the web view is accessible without logging in. If `nil` people have to log in before being able to do anything.

```
make-instance 'multi-site (&key cache-file prefix
  allow-prolog google-key allow-new-accounts-p
  anonymous-users-p allow-new-local-stores-p allow-new-
  remote-stores-p allow-new-empty-stores-p)
  → site
```

Sites of this type can serve multiple triple stores and allow users with the right permissions to create or open new stores. Most arguments have the same meaning they have for `site`, the others mean:

`allow-new-local-stores-p`
: When `t`, users with the `:site` permission can open new stores from local files. Defaults to `t`.

`allow-new-remote-stores-p`

As above, but for remote stores through a socket connection. Also defaults to `t`.

`allow-new-empty-stores-p`

Determines whether users can create new empty local stores. Defaults to `nil`.

`close-site (site)`

Close a site, releasing resources like open triple stores and AllegroCache databases.

`add-user (site name password &optional roles)`

Adds a new user to a site. `roles` can be a list of permissions this user has: `:site` for being allowed to configure the AllegroGraph Web View site, `:store` for being able to modify stores opened in read-write mode, and `:user` for being allowed to manage users.

`add-admin (site name password)`

Adds a user with all roles enabled.