LD Viewer - Manual

Denis Lukovnikov

June 2, 2014

The LD Viewer is a pure JavaScript application, built using AngularJS and Jassa (GeoKnow). However, to deploy the interface for your SPARQL endpoint and to customize it using Triple Actions, you do not need to know the specifics of neither. Settings configurations for some SPARQL point is explained in Section 3. Writing new Triple Actions to customize the interface is explained in Section 4. To extend the interface, one needs to be familiar with AngularJS concepts. In both cases, you would need to first install (Section 2) the interface somewhere for development.

1 Introduction

2 Install

To get the LD Viewer going on your installation, you would need to get the distribution files (JavaScript and CSS) and their dependencies. Then, these files need to be included in the `index.html`. To do so, one can simply clone the Github repository into the root folder of your server:

1. `git clone https://github.com/geoknow/ldviewer`
2. copy to the right folder or change config of your webserver

3 Configuration

By default, the interface is configured for the DBpedia dataset at `dbpedia.org/sparql`. To configure the interface for your endpoint, you need to change the settings of the `config.js` file in the `/src/dist/` folder and make it available for the application. The interface provides the following configuration options:

- `localgraph`: the RDF graph of which resources are displayed
- `endpoint`: the SPARQL endpoint where to get the data
- `owlgraph`: the list of RDF graphs from which information will be retrieved (corresponds to the `default-graph-uri` parameter of a SPARQL request.

- `primarylang`: the two-letter code for the default preferred language. Users will be able to override this setting.
- `fallbacklang`: the two-letter code of the language to use when information in the preferred language is not available. Users are not able to override this setting.
- `showLabels`: settings this to `true` will enable additional label fetching queries using the `labelPrefs` setting in order to show the labels of resources instead of prefixed URI's.
- `labelPrefs`: the array of label properties to be used to display as the title of resources. Predicates occurring earlier in the list are preferred over the later labels.
- `prefixes`: a hash of prefixes mapping from the partial URI to a prefix to represent the namespace. Prefixes are displayed when `showLabels` is disabled.
- `previewMappings`: an object describing which properties of a resources to get and where they will be displayed when the user hovers over a resource URI.

The `LDViewer.setConfig(setting, value)` function is used to set a certain setting to a certain value. For those familiar with AngularJS, these settings are stored in the `$rootScope`. To get the value of a set setting, the `LDViewer.getConfig(setting)` function can be used.

4 Customization

The configuration as described in Section 3 will only point the framework to the right dataset. To fully customize the framework for your data, you would need to write a few actions or customize existing ones. Triple Actions are used to populate several parts of the UI, mostly using hardcoded mappings from the retrieved subgraph for the viewed resource to interface elements. However, since different datasets may use different schema even for general information like resource label, description, etc..., these mappings need to be different for each one. In LD Viewer, these mappings are defined imperatively using Triple Actions. A Triple Action receives the values of a triple as arguments and can act upon this information. There are two kinds of Triple Actions, system actions and user actions. User actions provide additional interactivity in the triple table by allowing users to invoke the action on the triple by clicking on the action icon. System actions, however, are invisible to the users and can be used to populate interface elements.

The default collection of actions is defined in `taf.js` in the `src/config` directory. You can modify these actions or add your own. Triple Actions are written as objects that use Prototype's Class. Every Triple Action needs to subclass the `LDViewer.Action` class in order to be enabled.

Each Action is actually defined through its factory. The factory can have the following fields: `check()`, `legend()`, `action` (which contains the class of an actual Action) and `factory()`, the meanings of which will be explained later. To write a system action in its most basic form, one only needs to implement the `check()` method of ActionFactory. The Triple Action Framework (TAF) will pass three arguments to this `check()` function: the URI of the viewed resource, the predicate and the value. Note that for reverse relations, the URI of the viewed resource is still passed as the first argument even though it is the object of the triple. To find out whether the triple is reversed, one can simply check `predicate.forward`, which is set to `false` if the relation is reversed and the viewed resource is the object of the triple. For system actions, the `check()` function can contain all the logic of the system action, which will be executed when the triples are first loaded. In the current version of the TAF API, system actions and actions that should not be added as user actions next to a triple should return false when `check()` is called. For this reason, all system actions should throw an exception upon completion. Triple Actions should use the interface API exposed by `LDViewer`, which is described in Section 5.

5 Interface API

The global `LDViewer` variable exposes the API to manipulate the interface elements. Below follows a description of the different function provided by `LDViewer`. The functions are declared by different interface elements and will be grouped accordingly.

1. **Status**: this interface element can be used to tell the user what the system is doing. The Status element exposes only one function: `LDViewer.addStatus(status, icon)`

    - `LDViewer.addStatus(status, icon)`: status is a string that should be displayed next to the provided icon HTML code. The function returns an object with a `delete` method, which should be called when the added status is no longer relevant. For example, when loading triples, a status is added saying that data are being loaded. When the request from the endpoint returns, `delete` is called to no longer display the status saying that data are being loaded.

2. **Notifications**: displays notifications to the user. This element is currently used to say when something goes wrong.

    - `LDViewer.addNotification(notification, timeout)`: calling this results in a notification with the text provided by the `notification` string being displayed for `timeout` milliseconds.

3. **Shortcuts**: the shortcut box is used to provide shortcuts to frequently consulted properties of viewed resources and can be particularly useful when the viewed resource has a lot of information associated with it.

- `LDViewer.addShortcut(link, label, priority)`: adds a shortcut to the Shortcut interface element. `link` should be an anchor link, `label` will be displayed in the interface and `priority` indicates where in the list the shortcut will be displayed.

4. **Pretty Box**: is used to display a snippet of important information about the viewed resource.

   - `LDViewer.applyPrettyBox(fun)`: applies the function passed in the `fun` argument on the object describing the values in the Pretty Box. The `fun` function will be passed an object of which different values are used to populate different parts of the Pretty Box:
     - `thumbnail` should contain a list of Jassa URI nodes to images that are associated with the viewed resource. Only the first entry of thumbnail is shown to the user.
     - `label` should be a list of Jassa string literal nodes that contain the title of the viewed resource in different languages. Each language should only be present once.
     - `types` should be a list of Jassa resource nodes to a selection of resources that should be shown as types of the viewed resource.
     - `description` should be a list of Jassa string literal nodes that contain the descriptions of the viewed resource in different languages. Each language should only be present once.
     - `links` should be a hash from strings `label` to lists of Jassa URI nodes that point to external sources. The string `label` will be shown as the label the anchor text of a link. If the list of links for some `label` contains more than one entry, a dropdown will be used to group them under the `label` string. Otherwise, the `label` is ignored and the only link in the list is displayed directly. All links in the lists of links for all `label`s should be objects with two required fields: `uri` pointing to the destination and `plex` containing the string that should be used as the anchor text for the link.
     - `properties` defines the list of properties to be displayed in the lower part of the Pretty Box. However, the `LDViewer.getPrettyPropertyAdder()` function should be used to manipulate this list.
   - `LDViewer.getPrettyPropertyAdder(predicate, priority)`: this function should be used to add properties to the property table in the Pretty Box. This function must be provided with a Jassa resource node for `predicate` and a `priority` integer.
     Calling `LDViewer.getPrettyPropertyAdder()` returns a function that can be used to add values to the table. The returned function accepts one argument, a Jassa node object.

5. **Pretty Map** is an optional interface element that can be used to display coordinates on a map.

- `LDViewer.setMapCoord([latitude, longitude])` resets previously assigned map coordinates and uses the new coordinates to center the map on and to set a marker.