LD Viewer - Triple Action assignments

1 Tasks

Below follows a list of actions that could be implemented. If you are familiar with web development and JavaScript, these actions should not take more than a few hours of your time. However, the time needed may vary on the action specifications and your previous experience. So chose fast and wisely.

Alternatively, if you want to implement an interesting action that is not on the list, let us know.

1.1 Social integration

Enable sharing facts (represented by triples) on social networks. One idea is to share sentences like: “Did you know that...”.

1. Share the fact on Facebook
2. Share the fact on Twitter
3. Share the fact on G+

Tip: take into account different languages (see `primarylang` in the manual).

1.2 Search integration

1. write actions to search for the triple (or its parts) on Google, Yahoo! and Bing.

Tip: wrap the actions in action group(s) (see `ViewingGroup` in `src/dist/actions/taf.js`)

1.3 UI

Beware: these are system actions. System actions are not displayed as icons for the users to click on. Instead, they are executed when the data are loaded. Some say that system actions are easier to implement than user actions. For more on system actions, check the manual.

1. show birthplace/deathplace of a person on the map
    - Tip: take a look at the already implemented map action.

2. transform date displays to more intuitive presentations
    - Tip: take a look at Spotlight action.
3. transform number displays to more intuitive presentation
    - Tip: take a look at Spotlight action.

1.4 Random

1. notify user when viewed person is dead
2. notify user when viewed place is on the southern hemisphere

2 How to do it?

2.1 Steps

1. **fork it**: fork it on Github for the others to know what you're working on.
    - change the README.md in your fork so that it contains the description of the action you are going to implement on the first line - this should be no duplicate, so please first check all the forks of the repository before proceeding.
2. **clone it**: make a local clone of your forked github repository:
   `git clone http://github.com/__your_username__/ldviewer.git`
   and point your web server config to `index.html` in the root folder of the project.
3. **make it**: write your action in the template provided at `/dist/action.js`. See the manual and next section for more info. Remember to rename this file to avoid merge conflicts later on.
4. **push it**: make a pull request on Github.

3 How to write user actions

Take a look at the already implemented actions to get an idea how actions are written. The action definition is an object that defines a class using Class from PrototypeJS. The action definition for user actions can be seen as consisting of two parts. The first part defines the factory for the actions and contains

1. the `check()` method that is used to check whether the action is applicable to the triple.
2. the `legend` field describing what to display in the legend of the interface
3. the `action` field, which contains the definition of the action instance.

The definition of the action instance is the second part. The action instance definition contains

1. the `execute()` method which is executed when the user clicks on the action's icon in the interface

2. the `display()` function which should return the HTML that will be used to create the action icon in the interface. Note that the HTML code this function returns is sanitized by Angular's ngSanitize module, which may reject unsafe HTML.

3. the `description` field which should contain a short description of what this action does. This description will appear when the user hovers over the action icon in the interface.

Optionally, the factory object may also define a `factory` method. For more details, see the implementation of `LDViewer.ActionFactory` in `src/taf/core.js`.

All action factories MUST subclass `LDViewer.ActionFactory` and all action instance definitions should subclass `LDViewer.Action` using Class from PrototypeJS.

Finally, the action must explicitly be added to the interface by calling `LDViewer.taf.addAction()` or `Taf.addAction()` with the action factory object as the only argument.

Several actions may require additional support, for example, local/global action state and some actions may require using the LDViewer API. For the API, please check the manual. Please look at the already implemented actions to see how the API is used. An example of local action state is the Spotlight action (in `src/dist/actions/taf.js`), which remembers whether it is awaiting response from the Spotlight annotation endpoint. The Spotlight action also uses the Notification API and transforms the display values of triple values. Thus, the Spotlight action might be regarded as a reference implementation for several features of the Triple Action Framework. The notifications are useful for the last actions on our list. Transforming display values are useful for number and date formatting actions.