Delv Documentation

Release 1.0.0

Kris Zygmunt

October 11, 2013

CONTENTS

Delv – A framework for Dynamic Linked Visualizations

Delv is an interaction framework that will allow multiple visualizations written in languages such as Processing and D3.js to communicate and interact with each other. Be able to combine multiple views of your data into one powerful, insightful visualization.

CHAPTER

ONE

INTRODUCING DELV

CHAPTER

TWO

DELV EXAMPLES

2.1 inSite

To run the inSite example in Processing, open `/your/path/to/delv/examples/inSite/inSite.pde` from your Processing environment and press play. To see the same example in Javascript, point a web browser at `file:///your/path/to/delv/examples/inSite/inSite.xhtml`. This is an example of several interactions including category filtering, an interactive color choosing legend, rollover / hovering, etc.

CHAPTER

THREE

API REFERENCE

3.1 DataIF API

Delv defines a common abstract data interface that views can use to access datasets. This interface allows the view to access data without needing to know or understand where or how the data is stored. DataIF signals:

- categoryVisibilityChanged
- categoryColorsChanged
- hoveredCategoryChanged
- highlightedCategoryChanged
- selectedIdsChanged
- highlightedIdChanged
- hoveredIdChanged

class dataIF
the abstract data interface

dataIF.**updateCategoryVisibility**(*invoker*, *dataset*, *attribute*, *selection*)

Parameters

- **invoker** (*string*) – The name of the object that is toggling the visibility
- **dataset** (*string*) – The name of the dataset whose attribute's category's visibility is toggling
- **attribute** (*string*) – The name of the attribute whose category's visibility is being toggled
- **selection** (*string*) – The name of the category whose visibility is toggling

toggle the visibility of the category specified by selection. Any item whose attribute's value matches category will have it's visibility toggled.

dataIF.**updateHighlightedId**(*invoker*, *dataset*, *id*)

Parameters

- **invoker** (*string*) – The name of the object that is updating the highlighted id
- **dataset** (*string*) – The name of the dataset whose identifier is being highlighted
- **id** (*string*) – The id of the item that is being highlighted

updateHighlightedId sets which item specified by id is highlighted. To have no item highlighted, use "" for the id.

dataIF.**updateHoveredId**(*invoker*, *dataset*, *id*)

Parameters

- **invoker** (*string*) – The name of the object that is updating the hovered id
- **dataset** (*string*) – The name of the dataset whose identifier is being hovered
- **id** (*string*) – The id of the item that is being hovered

updateHoveredId sets which item specified by id is hovered. To have no item hovered, use "" for the id.

`dataIF.`**`getAllItems`**(*dataset*, *attribute*)

Parameters

- **dataset** (*string*) – The name of the dataset that has the attribute
- **attribute** (*string*) – The name of the attribute whose items should be returned

Returns Returns a String[] list of all the items for that attribute

Get all the items (as Strings) for attribute from dataset. Returns an empty String[] on error.

`dataIF.`**`getAllIds`**(*dataset*, *attribute*)

Parameters

- **dataset** (*string*) – The name of the dataset that has the attribute
- **attribute** (*string*) – The name of the attribute whose iss should be returned

Returns Returns a String[] list of all the ids of the items for that attribute

Get all the ids of the items (as Strings) for attribute from dataset. Returns an empty String[] on error.

`dataIF.`**`getItem`**(*dataset*, *attribute*, *identifier*)

Parameters

- **dataset** (*string*) – The name of the dataset that has the attribute
- **attribute** (*string*) – The name of the attribute whose item should be returned
- **identifier** (*string*) – The id of the item to return

Returns Returns a String representation of the item with this id for that attribute

Get the item (as a String) for attribute from dataset. Returns an empty String on error.

`dataIF.`**`getHighlightedId`**(*dataset*)

Parameters **dataset** (*string*) – The name of the dataset

Returns Returns the id of the item in the dataset that is currently highlighted

Get the id of the currently highlighted item in the dataset. If none is highlighted, returns an empty string.

`dataIF.`**`getHoveredId`**(*dataset*)

Parameters **dataset** (*string*) – The name of the dataset

Returns Returns the id of the item in the dataset that is currently hovered

Get the id of the currently hovered item in the dataset. If none is hovered, returns an empty string.

3.2 Delv API

Delv signals:

- categoryVisibilityChanged
- hoveredIdChanged

delv

The delv interface provides signal handling capabilities for the delv signals and other custom signals that are registered with it.

`delv.`**`log`**(*msg*)

Parameters **log** (*string*) – a message to log

In javascript, this will print the msg to the console. In Processing, this is a call to println().

`delv.`**`getDataIF`**(*id*)

Parameters **id** (*string*) – name of the requested dataIF instance

Returns a reference to a `dataIF` (page 7) instance

returns the reference to the requested dataIF instance.

`delv.`**`emitSignal`**(*signal*, *invoker*, *dataset*, *attribute*)

Parameters

- **signal** (*string*) – The name of the signal to emit
- **invoker** (*string*) – The name of the object emitting the signal
- **dataset** (*string*) – The name of the dataset pertaining to the reason the signal is emitted
- **attribute** (*string*) – The name of the attribute pertaining to the reason the signal is emitted

emitSignal is a way for either the data interface to send signals to delv or for views to send custom, unrecognized signals through delv to the other views. This is a good mechanism for sending signals that relate to changing features of the visualization unrelated to the data. For instance, to change paragraph alignment, an alignment selector view might send an `alignmentChanged` signal as follows:

```
delv.emitSignal("alignmentChanged", "alignmentView", "", "LEFT");
```

where the dataset is left blank since it is not associated with a particular data set.

`delv.`**`connectToSignal`**(*signal*, *name*, *method*)

Parameters

- **signal** (*string*) – The name of the signal, can be one of the delv *signals* (page 9), or a custom signal
- **name** (*string*) – The name of the view whose method should be invoked when the signal has been emitted
- **method** (*string*) – The name of the method which should be called when the signal has been emitted

connectToSignal registers the view's name and method with delv's signal handler. Whenever the signal has been emitted, delv will invoke the registered methods as `name.method(`*`invoker, dataset, attribute`*`);` where invoker is the name of the object that emitted the signal, dataset is the name of the data set associated with the signal, and attribute is the name of the data attribute associated with the signal.

To unregister a method from delv's signal handler, use `delv.disconnectFromSignal()` (page 10)

`delv.`**`disconnectFromSignal`**(*signal*, *name*)

Parameters

- **signal** (*string*) – The name of the signal to disconnect from
- **name** (*string*) – The name of the view whose method should no longer be invoked when the signal is emitted.

disconnectFromSignal unregisters a view's method from the delv's signal handler for the given signal.

Note: Only one method per view can be registered with a particular signal in delv.

`delv.`**`addView`**(*view*, *id*)

Parameters

- **view** – the view instance to be added to Delv's list of known views
- **id** (*string*) – a string uniquely identifying this view instance

Returns the delv instance (to support method chaining)

Registers a view with Delv. `delv.d3Chart` (page 11) and `delv.processingSketch` (page 11) both automatically register the view they construct with Delv, so this method **does not** need to be called for views constructed in this fashion.

Supports method chaining.

`delv.`**`reloadData`**(*source*)

Parameters **source** – a string identifying who is calling reloadData

Delv will tell all views to reloadData(source). Call this when a data source has finished initial load or has changed.

3.3 delv.js API

`delv`

delv is a namespace implementing the delv interface that is created when delv.js is loaded. In addition to the basic delv API, delv.js also provides some helper functions to support using delv in an asynchronous environment and to ease the integration of visualizations written in Processing or D3.js.

`delv.`**`addDataIF`**(*dataIF*, *id*)

Parameters

- **dataIF** – reference to a `dataIF` (page 7) instance
- **id** (*string*) – unique name of the dataIF instance

addDataIF adds the dataIF to Delv's list of dataSources and also connects the dataIF's *signals* (page 7) to Delv's signal handler.

Supports method chaining.

delv.**resizeAll**()
Has delv resize all views based on the current window size. Delv automatically registers resizeAll to listen for the SVGResize and resize events, so you should not need to call resizeAll yourself.

---

class delv.**d3Chart**(*elementId*, *script*, *viewConstructor*, *loadCompleteCallback*)

Parameters

- **elementId** (*string*) – ID of the html document element where the d3Chart should be drawn to.
- **script** (*string*) – name of the javascript file containing the code implementing the view and view for this chart
- **viewConstructor** (*string*) – name of the view constructor method that should be used to construct the view that will be drawing in this chart
- **loadCompleteCallback** (*function*) – function to call when the script has finished loading and the view has been successfully constructed

d3Chart is a helper class that loads the given script on demand, constructs the specified view and tells it to draw in the given html element, and finally calls the loadCompleteCallback once all of this has been done successfully.

---

class delv.**processingSketch**(*canvas*, *sketchList*, *viewConstructor*, *loadCompleteCallback*)

Parameters

- **canvas** (*string*) – ID of the html5 canvas element where the processing sketch should draw to.
- **sketchList** (*string*) – list of the processing .pde files containing the code implementing the view, view and helper objects for this sketch
- **viewConstructor** (*string*) – name of the view constructor method that should be used to construct the view that will be drawing this sketch
- **loadCompleteCallback** (*function*) – function to call when the sketches have finished loading and the view has been successfully constructed

processingSketch is a helper class that loads the given processing .pde files on demand, constructs the specified view and tells it to draw in the given html5 canvas element, and finally calls the loadCompleteCallback once all of this has been done successfully.

Note: Be sure to include DelvView.pde, Delv1DView.pde and Delv.pde in the sketchList

---

class delv.**view**

view.**dataIF**(*name*)

Parameters **name** (*string*) – The name of the data interface

sets the view's DataIF to the data interface with the given name

view.**getName**()
view.**setName**(*name*)
Get and set the name of the view.

Supports method chaining

view.**connectSignals**()
The default implementation has no signals connected. This method is called by Delv once the view has been fully loaded in Javascript. This is the recommended place to put calls to delv.connectToSignal() (page 9) like:

```
_delvIF.connectToSignal("categoryVisibilityChanged", "YourViewName", "YourVisibilityChangedCallback");
_delvIF.connectToSignal("YourCustomSignal", "YourViewName", "YourCustomSignalCallback");
```

See also:

See *delv signals* (page 9) for a complete list of recognized Delv signals.

---

class delv.**d3View**(*svgElem*)

Parameters **svgElem** (*string*) – the name of the svg element that the d3 view can modify and draw in

the d3 view is a delv.view (page 11) that additionally knows the name of the containing svg element that it is allowed to modify and draw in

---

class delv.**d3HierarchyView**(*svgElem*)

Parameters **svgElem** (*string*) – the name of the svg element that the d3 view can modify and draw in

the d3 hierarchy view is a delv.d3View (page 12) that knows how to convert attributes defining node size, node name, link start, and link end into the hierarchy data object recognized by d3 examples such as bar_hierarchy, force_collapsible, partition_sunburst_zoom, and tree_interactive

d3HierarchyView.**getNodeDatasetName**()
d3HierarchyView.**setNodeDatasetName**(*name*)
Get or set the name of the dataset containing the node name and node size attributes.

Supports method chaining.

d3HierarchyView.**getLinkDatasetName**()
d3HierarchyView.**setLinkDatasetName**(*name*)
Get or set the name of the dataset containing the link start and link end attributes.

Supports method chaining.

d3HierarchyView.**getNodeSizeAttr**()
d3HierarchyView.**setNodeSizeAttr**(*name*)
Get or set the name of the attribute that describes the node size.

Supports method chaining.

d3HierarchyView.**getNodeNameAttr**()
d3HierarchyView.**setNodeNameAttr**(*name*)
Get or set the name of the attribute that contains the unique identifiers for the nodes.

Supports method chaining.

d3HierarchyView.**getLinkStartAttr**()
d3HierarchyView.**setLinkStartAttr**(*name*)
Get or set the name of the attribute that contains the names of the nodes that are the start of each link.

Supports method chaining.

d3HierarchyView.**getLinkEndAttr**()

d3HierarchyView.**setLinkEndAttr**(*name*)

Get or set the name of the attribute that contains the names of the nodes that are the end of each link.

Supports method chaining.

d3HierarchyView.**convertToHierarchy**()

Returns the node hierarchy in a data object that the d3 hierarchy views can accept as input to their bindData method

Convert from node size, node name, link start and link end lists into a hierarchy of objects with name, tag, children, and size attributes.

CHAPTER

FOUR

USING DELV WITH PROCESSING

4.1 Processing project layout

For a standard Delv Processing project, the following directory structure is recommended:

- `your_project_directory`
  - `your_app.xhtml` – See *xhtml Layout Templates* (page 19) for a layout template.
  - `javascript` – In order to work in the web browser, the javascript code should be in a subdirectory relative to your xhtml.
    - `your_javascript_app.js` – This file contains the logic to construct and connect the visualizations together, including configuration of the views. See *Javascript Templates* (page 22) for an example of how to setup Delv and configure views.
    - `delv.js`
    - `jquery-1.7.2.js`
    - `processing-1.4.0.js`
  - `your_processing_sketch` – to have this sketch run in Processing as well as in the web browser, make sure that you have a corresponding .pde file in this directory with the same name
    - `your_processing_sketch.pde` – main entry to your processing sketch. [1]
    - `your_processing_vis.pde` – This file contains the view for one visualization type. See *Processing Templates* (page 25) for an example View. [2]
    - `your_other_processing_vis.pde` – This file contains the view for another visualization type. [2]
    - `Delv.pde` [3]
    - `DelvBasicData.pde` [3]
    - `DelvView.pde` [3]

[1] Usually this main sketch is only used in the Processing environment, and should **NOT** be loaded by your_javascript_app.js

[2] Make sure all files including Delv.pde, DelvBasicData.pde and DelvView.pde are explicitly loaded in your_javascript_app.js

[3] For the web browser, this file could be in another location, but to work in Processing it needs to be in the same directory of your sketch (you can use a link if you don't want to copy the file).

CHAPTER

FIVE

USING DELV WITH D3.JS

5.1 D3.js project layout

For a standard Delv d3.js project, the following directory structure is recommended:

- `your_project_directory`
  - `your_app.xhtml` – See *xhtml Layout Templates* (page 19) for a layout template
  - `javascript` – In order to work in the web browser, the javascript code should be in a subdirectory relative to your xhtml.
    - `your_javascript_app.js` – This file contains the logic to construct and connect the visualizations together, including configuration of the views. See *Javascript Templates* (page 22) for an example of how to setup Delv and configure views.
    - `delv.js`
    - `jquery-1.7.2.js`
    - `d3.v2.js`
    - `your_d3_chart_type.js` – This file contains the view for one d3 chart type. See *D3 Templates* (page 27) for an example view.
    - `your_other_d3_chart_type.js` – This file contains the view for another d3 chart type.

CHAPTER

SIX

CODE TEMPLATES

Note: These templates assume that the files are organized in the same way as this *processing project layout* (page 15) or this *d3.js project layout* (page 17). Adjust the templates to match your file organization.

6.1 xhtml Layout Templates

See also:

Download `example xhtml layout`

```
<!DOCTYPE html>
<html
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/1999/xhtml"
   xml:lang="en"
   lang="en">
<head>

<!-- Change the title to something appropriate for your website / -->
<!-- application -->
<title>Example App</title>

<!-- delv uses jquery -->
<script type="text/ecmascript" id="jquery.script" src="./javascript/jquery-1.7.2.js"></script>

<!-- include d3 if using d3.js -->
<script type="text/ecmascript" id="d3.script" src="./javascript/d3.v2.js"></script>

<!-- include processing if using processing.js -->
<script type="text/ecmascript" id="processing.script" src="./javascript/processing-1.4.0.js"></script

<!-- include delv -->
<script type="text/ecmascript" id="delv.script" src="./javascript/delv.js"></script>

<!-- script containing the application-specific logic for connecting -->
<!-- to delv and configuring mediators. -->
<script type="text/ecmascript" id="app.script" src="./javascript/example_app.js"></script>
</head>

<!-- Note that onload calls init() which should be defined in your app -->
<!-- script.  Also, the body onresize should call delv.resizeAll() -->
```

```
<!-- (or a custom function that also calls delv.resizeAll()) so that -->
<!-- each mediator/view can resize and redraw itself appropriately.-->
<body onload="init()" onresize="delv.resizeAll()">

<!-- Note that sizes are expressed as percentages in most places.  It -->
<!-- is possible to use pixels for sizing, at least for individual -->
<!-- divs within the layout. -->
<div
  id="YourApp"
  style="overflow:hidden;zindex:1;top:0%;left:0;width:100%;height:100%;position:absolute">

  <!-- create a layout with 4 d3.js views and 1 processing view -->
  <!-- ____________________________________________________________ -->
  <!-- |                            |                             | -->
  <!-- |       d3 view 1            |        d3 view 2            | -->
  <!-- |                            |                             | -->
  <!-- ____________________________________________________________ -->
  <!-- |                            |                             | -->
  <!-- |       d3 view 3            |        d3 view 4            | -->
  <!-- |                            |                             | -->
  <!-- ____________________________________________________________ -->
  <!-- |                     processing view                      | -->
  <!-- ____________________________________________________________ -->
  <!-- use style on the div to create the desired layout, then tell -->
  <!-- the svg elements to use 100% of the space they've been -->
  <!-- allocated -->

  <div
    id = "d3_views"
    style="overflow:hidden;zindex:1;top:0%;left:0;width:100%;height:90%;position:absolute">
    <!-- Note that the d3 views are svg elements -->
 <div
    id ="d3_view1_container"
    style="overflow:hidden;zindex:3;top:3%;left:3%;width:47%;height:47%;position:absolute">
    <!-- if id matches the name of the mediator for this element, then -->
    <!-- the example_app.js can use this assumption to automatically -->
    <!-- construct the mediator based on element id.  Otherwise, will -->
    <!-- need to pass in the name of the desired mediator to a -->
    <!-- javascript construction function of some sort. -->
    <!-- In this case, construct a partition_sunburst_zoom_mediator -->
    <svg:svg
       id="partition_sunburst_zoom"
       class="d3"
       x="0%"
       y="0%"
       width="100%"
       height="100%"
       version="1.1"
       style="overflow:hidden;zindex:3;position:absolute">
   </svg:svg>
  </div>
  <div
    id ="d3_view2_container"
    style="overflow:hidden;zindex:3;top:3%;left:50%;width:47%;height:47%;position:absolute">
    <svg:svg
       id="bar_hierarchy"
       class="d3"
       x="0%"
```

```
        y="0%"
        width="100%"
        height="100%"
        version="1.1"
        style="overflow:hidden;zindex:3;position:absolute">
      </svg:svg>
    </div>
    <div
      id ="d3_view3_container"
      style="overflow:hidden;zindex:3;top:50%;left:3%;width:47%;height:47%;position:absolute">
      <svg:svg
        id="force_collapsible"
        class="d3"
        x="0%"
        y="0%"
        width="100%"
        height="100%"
        version="1.1"
        style="overflow:hidden;zindex:3;position:absolute">
    </svg:svg>
    </div>
    <div
      id ="d3_view4_container"
      style="overflow:hidden;zindex:3;top:50%;left:50%;width:47%;height:47%;position:absolute">
      <svg:svg
        id="tree_interactive"
        class="d3"
        x="0%"
        y="0%"
        width="100%"
        height="100%"
        version="1.1"
        style="overflow:hidden;zindex:3;position:absolute">
     </svg:svg>
    </div>
    </div>
      <!-- Note that the processing views are canvas elements -->
      <div
      id = "processing_views"
      style="overflow:hidden;zindex:1;top:90%;left:0;width:100%;height:10%;position:absolute">
      <!-- if id matches the name of the mediator for this element, then -->
      <!-- the example_app.js can use this assumption to automatically -->
      <!-- construct the mediator based on element id.  Otherwise, will -->
      <!-- need to pass in the name of the desired mediator to a -->
      <!-- javascript construction function of some sort. -->
      <!-- In this case, construct a RegionMediator -->
      <div
        id="processing_view_container"
        style="overflow:hidden;zindex:3;top:2.5%;left:2.5%;width:95%;height:95%;position:absolute">
        <canvas
        id="Region"/>
      </div>

        </div>
</div>
</body>
</html>
```

6.2 Javascript Templates

See also:

Download `example javascript app utilities`

```
// ========================================================================
// Copyright (c) 2013, Scientific Computing and Imaging Institute,
// University of Utah. All rights reserved.
// Author: Kris Zygmunt
// License: New BSD 3-Clause (see accompanying LICENSE file for details)
// ========================================================================

function init() {
  console.log("entering init");
  dataLoaded = false;
  d3Array = Array.prototype.slice.call(document.getElementsByTagNameNS("*","svg"));
  for (var j = 0; j < d3Array.length; j++) {
    var id = d3Array[j].getAttribute("id");
    var svgElem = d3Array[j];
    console.log("loading script: " + "./" + id + ".js");
    var chart = new delv.d3Chart(id, "./"+id+".js", "d3WrapperNS."+id+"_view", init_view_instance);
  }

  canvasArray = Array.prototype.slice.call(document.getElementsByTagNameNS("*","canvas"));
  for (var j = 0; j < canvasArray.length; j++) {
    var id = canvasArray[j].getAttribute("id");
    var canvas = canvasArray[j];
    var sketch = new delv.processingSketch(canvas,
                                           ["./Globals.pde",
                                            "./Attribute.pde",
                                            "./DelvView.pde",
                                                  "./Delv1DView.pde",
                                                  "./Delv2DView.pde",
                                                  "./DelvCategoryView.pde",
                                            "./Delv.pde",
                                            "./BasicRegion.pde", // currently only needed for the Re
                                            "./"+id+".pde"],
                                           id+"View",
                                           init_view_instance);
  }

  // TODO just a demonstration of how to send a signal back to Python
  $( document ).bind("testEventForPython", emitPythonEvent);

  // TODO to get behavior working like releasing a scroll bar while not over a particular view,
  // may need to catch the document-level mouse released event and forward it on to all views
  // otherwise when mouse is moved back into that view, the view will be behaving as if the mouse is

  if (typeof(dataIF) === "undefined") {
    // dataCanvasId = "d3DemoData"
    // var canvas = document.createElement('canvas');
    // canvas.id     = dataCanvasId;
    // canvas.width  = 0;
    // canvas.height = 0;
    // canvas.style.zIndex   = 8;
    // document.body.appendChild(canvas);
```

```
    // Processing.loadSketchFromSources(dataCanvasId,
                //                          ["./Globals.pde",
                //                                 "./DelvBasicData.pde",
                //                                 "./DelvColorMap.pde",
                //                                 "./DelvEnums.java",
                //                           "./d3DemoData.pde"]);
          // setTimeout(finishLoadingData, 50);
    dataIF = new d3WrapperNS.d3_demo_data("d3Demo");
    dataIF.load_data();
    delv.giveDataIFToViews("d3Demo");
    delv.reloadData("d3Demo");

  } else {
    delv.addDataIF(dataIF, "d3Demo");
  }

  resizeAll();
}

function finishLoadingData() {
  p = Processing.getInstanceById(dataCanvasId);
  try {
          pDataIF = new p.d3DemoData();
          dataLoaded = true;
          delv.log("Test data initialized!!!");
  } catch (e) {
          delv.log("initializing Test data failed.  Try again later");
          setTimeout(finishLoadingData, 5);
          dataLoaded = false;
  }
  if (dataLoaded) {
    pDataIF.loadData();
    pDataIF.setDelvIF(delv);
    delv.addDataIF(pDataIF, "d3Demo");
    delv.giveDataIFToViews("d3Demo");
    delv.reloadData("d3Demo");
  }
}
function init_view_instance(view, elemId) {
  delv.log("init_view_instance(" + view + ", " + elemId + ")");
  view.dataIF("d3Demo");
  if (elemId == "Region") {
  delv.log("init_view_instance Region setup");
    view.name("d3Demo.Region");
    aboveDataset = view.createDataset("Nodes");
    aboveDataset.barStartAttr("size")
                .barTagAttr("name")
                .units("loc")
                .defaultRegionType("allNodes")
                .defaultBarHeight("1.0")
                .defaultBarLength("0")
                .defaultBarType("node");
    view.addDataset(aboveDataset, false);

  } else if (elemId == "partition_sunburst_zoom") {
    view.setName("d3Demo.partition_sunburst_zoom")
            .setNodeDatasetName("Nodes")
            .setLinkDatasetName("Links")
```

```
            .setNodeSizeAttr("size")
            .setNodeNameAttr("name")
            .setLinkStartAttr("StartNode")
            .setLinkEndAttr("EndNode");

  } else if (elemId == "force_collapsible") {
    view.setName("d3Demo.force_collapsible")
            .setNodeDatasetName("Nodes")
            .setLinkDatasetName("Links")
            .setNodeSizeAttr("size")
            .setNodeNameAttr("name")
            .setLinkStartAttr("StartNode")
            .setLinkEndAttr("EndNode");

  } else if (elemId == "bar_hierarchy") {
    view.setName("d3Demo.bar_hierarchy")
            .setNodeDatasetName("Nodes")
            .setLinkDatasetName("Links")
            .setNodeSizeAttr("size")
            .setNodeNameAttr("name")
            .setLinkStartAttr("StartNode")
            .setLinkEndAttr("EndNode");

  } else if (elemId == "tree_interactive") {
    view.setName("d3Demo.tree_interactive")
            .setNodeDatasetName("Nodes")
            .setLinkDatasetName("Links")
            .setNodeSizeAttr("size")
            .setNodeNameAttr("name")
            .setLinkStartAttr("StartNode")
            .setLinkEndAttr("EndNode");
  }
  // if (dataLoaded) {
    delv.log("init_view_instance reloadData");
    view.reloadData("d3Demo.js");
  // }
  delv.log("init_view_instance resizeAll");
  resizeAll();
}

function logEvent(evt) {
  console.log("Event " + evt + " logged.");
}

function resizeAll() {
  delv.resizeAll();
}

// kept around in case we need to do this in the future, but not used really right now
function emitPythonEvent(e, name, args) {
  if (typeof(QtWin) != "undefined") {
    if (typeof(e) === "object") {
      console.log("QtWin: " + QtWin + ", event(" + typeof(e) + "): " + name);
      QtWin.emitEvent(name, args);
    }
    else {
      console.log("QtWin: " + QtWin + ", event(" + typeof(e) + "): " + e.toString());
      QtWin.emitEvent(e.toString(), args);
```

```
    }
  }
  else {
    console.log("No QtWin yet");
  }
}
```

6.3 Processing Templates

See also:

Download `example processing template`

```
////////////////////////////////////
//              View              //
////////////////////////////////////

// EDIT: search and replace YourView with the name of your view

// NOTE: extend DelvBasicView which implements DelvView to get some common delv and processing.js fu

class YourView extends DelvCategoryView {
  // EDIT: add any view member variables
  // String[] choices;
  // boolean[] choices_selected;

  YourView() {
    // EDIT: provide a more appropriate default name for the view
    this("DefaultName");
  }

  YourView(String name) {
    // NOTE: be sure to call the superclass's constructor
    super(name);
    // EDIT: any other appropriate initialization
    choices = new String[0];
    choices_selected = new boolean[0];
    textFont( _verdana_font_12 );
  }

  // EDIT: add any other getters and setters for view variables

  // EDIT: implement the methods that the will pass data to be visualized
  void setChoices(String[] new_choices) {
    choices = new_choices;
    choices_selected = new boolean[choices.length];
    for (int i = 0; i < choices.length; ++i) {
      choices_selected[i] = true;
    }
    // NOTE: when this is used in processing.js, the normal draw cycle doesn't happen as expected.
    // therefore, you should call redraw() explicitly when the data or visualization has changed.
    redraw();
  }

  void setSelectedChoices(String[] selected_choices) {
    for (int i = 0; i < choices.length; ++i) {
```

```
      choices_selected[i] = false;
      for (int j = 0; j < selected_choices.length; ++j) {
        if (selected_choices[j].equals(choices[i])) {
          choices_selected[i] = true;
        }
      }
    }
    // NOTE: when this is used in processing.js, the normal draw cycle doesn't happen as expected.
    // therefore, you should call redraw() explicitly when the data or visualization has changed.
    redraw();
  }

  // NOTE: The parent class View's draw() method takes care of setting the background
  // and translating to _origin.
  // After the translation, draw will call render() which should be overridden here.
  // After render() has completed, draw will translate back from _origin.
  void render() {
    // EDIT: draw stuff here!
  }

  void setup() {
    // EDIT: implement as you would any Processing setup method
  }

  // EDIT: The parent class implements resize by setting _w and _h, and calling redraw()
  // reimplement resize here if this is insufficient for your purposes.

  void mouseMovedInView(int mx, int my) {
    // EDIT: handle mouse move events
    // NOTE: remember to call redraw() explicitly to ensure that the visualization is updated
    // for processing.js
    // NOTE: remember to notify DelvBasicView if some event has happened (like a hover has occurred)
    // so that the View can notify the world.
  }

  void mousePressedInView(int mx, int my, boolean rightPressed) {
    // EDIT: handle mouse press events
    // NOTE: remember to call redraw() explicitly to ensure that the visualization is updated
    // for processing.js
    // NOTE: remember to notify DelvBasicView if some event has happened (like a hover has occurred)
    // so that the View can notify the world.
  }

  void mouseReleasedInView(int mx, int my) {
    // EDIT: handle mouse release events
    // NOTE: remember to call draw() explicitly to ensure that the visualization is updated
    // for processing.js
    // NOTE: remember to notify DelvBasicView if some event has happened (like a hover has occurred)
    // so that the View can notify the world.
  }

  void mouseDraggedInView(int mx, int my) {
    // EDIT: handle mouse drag events
    // NOTE: remember to call draw() explicitly to ensure that the visualization is updated
    // for processing.js
    // NOTE: remember to notify DelvBasicView if some event has happened (like a hover has occurred)
    // so that the View can notify the world.
  }
```

```
  void mouseScrolledInView(int wr) {
    // EDIT: handle mouse scroll events
    // NOTE: remember to call draw() explicitly to ensure that the visualization is updated
    // for processing.js
    // NOTE: remember to notify DelvBasicView if some event has happened (like a hover has occurred)
    // so that the View can notify the world.
  }

}
```

6.4 D3 Templates

- *genindex*
- *search*