Exhibit 3.0 Documentation

Welcome to Exhibit 3.0. This documentation is also available on the Exhibit 3.0 documentation wiki at http://www.simile-widgets.org/wiki/Exhibit3 where you can find and update the content for the life of the project.

Table of Contents

Exhibit Developer Documentation

See the developer documentation for Exhibit developers who want to customize or examine how Exhibit code works.

Developer Doc for Exhibit 3.0:
https://github.com/zepheira/exhibit3/wiki

Developer Doc for Backstage and Exhibit 3.0 Staged Mode:
https://github.com/zepheira/backstage/wiki

Exhibit 3.0

Publishing Framework for Large-Scale Data-Rich Interactive Web Pages

Exhibit 3.0 is a powerful yet easy-to-use open source publishing framework for large-scale data-rich interactive web pages.

Building on the success of the original Exhibit, version 3.0 lets you easily create web pages to publish and visualize data collections ranging from small personal collections in Scripted mode, up to large data sets in the server-based Staged mode.

- **Exhibit 3.0 Scripted (rc1):** With Exhibit 3.0 Scripted mode, you can visualize data in a Web browser with a simple HTML-based configuration. No programming or server-side set up required.
- **Exhibit 3.0 Staged (beta2):** Staged mode requires the use of server software to publish bigger data sets.

What is Exhibit 3.0?

Exhibit 3.0 lets you publish data-rich web pages without complicated programming:

- Free, no cost
- Easy to use – No programming skills required
- Open source platform – Get involved, share your expertise, write code or add a demo
- Scalability – Staged mode scales to hundreds of thousands of items
- Lightweight publishing framework for building interactive web pages of linked data
- Supports search (Scripted mode), faceted navigation, interactive displays
- Easy to reconfigure and extend
- Supports customized data display

Scripted Mode for Publishing Smaller Data Sets in the Browser

Exhibit 3.0 Scripted is designed for smaller data sets – for publishing rich interactive exhibits, with thousands of items, right in your Web browser.

Download + Install | User Documentation and Tutorial | Developer Documentation

Staged Mode Offers Scalability with Simplicity

Exhibit 3.0 Staged mode extends the capacity of Exhibit by combining the in-browser software with greater capacity of a server-based component. The server stores and indexes data, and handles browser queries.

Download + Install | Developer Documentation

Getting Involved in the Exhibit Community

As an open source project, Exhibit relies on code and ideas from the community. Meet other exhibit users and developers on IRC on freenode, or browse the SIMILE Widgets mailing list archives to ask or answer questions with other Exhibit users. Chances are others may have similar questions, and the list is a great place to share answers.

What's New in Exhibit 3.0?

Exhibit 3.0 offers a powerful, easy-to-use publishing framework for building large-scale, data-rich, interactive web pages.

Building on the success of Exhibit 2, the new version offers two modes:

Exhibit 3.0 Scripted (rc1) for smaller in-browser data sets

Exhibit 3.0 Staged (beta) built on Backstage, for much bigger server-based data sets

Exhibit remains free, open source software that's easy to use. Highlights of the new release include:

Exhibit 3.0 Scripted (rc1):

- Browse thousands of items
- New HTML5 configuration language, with backwards compatibility for prior configuration language
- New localization system (no more 404's)
- New history system (no more __history__.html)
- Updated libraries, including jQuery
- Removal of external or unconfigurable service dependencies within the core
- Implements most core views and facets and some popular extension views
- More developer friendly, including a new event-driven API, basic tests, and documentation
- Persistence: Pick up where you left off browsing an Exhibit
- Bookmarks: Share exactly what you see with others

Exhibit 3.0 Staged (beta2):

- Scalability: Browse hundreds of thousands of items
- Persistence: Pick up where you left off browsing an Exhibit
- Export data in HTML + RDFa format

This section explains how to choose which mode of Exhibit 3.0 is right for you and maps feature differences between Exhibit 2 and Exhibit 3.0.

Choosing Between Exhibit 3.0 Scripted and Staged

This section explains some of the differences between the Scripted and Staged modes of Exhibit to help you choose which is right for you.

Generally speaking, the size of your data set will determine whether you choose to use the **Scripted** or **Staged** mode of Exhibit 3.0.

Exhibit 3.0 Scripted: In-Browser Scripts

Smaller data sets numbering a few dozen, a few hundred, or up to a thousand items can run in the browser using the Scripted mode of Exhibit. There is no set item limit for Scripted mode. If your data set has smaller items with few properties and short values, you may find Scripted mode handles a few thousand smaller items.

No programming is required beyond the basic HTML you use to author an Exhibit page.

Exhibit 3.0 Staged: Server-Based

Larger data sets – up to hundreds of thousands of items – are better suited to Staged mode.

Running Staged mode requires that you host the server software yourself or locate a provider who can host it for you.

Differences in Functionality: Scripted vs. Staged Mode

The Scripted and Staged modes share some of the same features. Differences in functionality are listed below. Differences in functionality between Scripted and Staged mode may affect your choice of which mode is best for your application.

Note: Features not listed here do not carry over from Scripted to Staged.

Views

Exhibit 3.0 Staged includes the Tile View, but it will only show the first twenty items at a time. None of the controls for sorting fields, sort order, or grouping are available. A pagination feature is expected to be developed soon.

Facets

Exhibit 3.0 Staged implements the List Facet, but any facet value with a count of one will be omitted.

Lens Language

Exhibit 3.0 Staged implements a subset of the lens language and recognizes the following lens attributes:

- `if-exists`
- `*-content`
- `*-subcontent`
- `*-style-content`
- `*-style-subcontent`

Expression Language

Some of the Exhibit expression language is implemented in Exhibit 3.0 Staged:

- Forward segments (`.`)
- Backward segments (`!`)
- `if`
- `if-exists`

Collections

Exhibit 3.0 Staged allows for subsets of the contents of its database to be grouped into collections, based on the following divisions:

- All items (default)
- Only items of one certain `rdf:type`

To learn more about Exhibit 3.0 Staged, see the Exhibit developer documentation on GitHub.

Feature Map: What's Supported in Exhibit 2 and Exhibit 3.0?

For existing Exhibit users and developers, the following table compares the features supported in Exhibit 3.0 and the previous release, Exhibit 2.2.0. This list will be updated as new features are developed.

Some of the features are marked as "no plans to implement" which suggests members of the Exhibit community may want to develop and deploy them as Exhibit 3.0 extensions. They are not likely to be included in the core of Exhibit 3.0, but discussion is welcome. Join the conversation on the SIMILE Widgets mailing list.

General Notes for Exhibit 2 Users

The expression language remains the same between Exhibit 2 and Exhibit 3.0.

Once you've installed and set up the new Exhibit 3.0 scripts on your Web server, your existing exhibits should display without modification. See the notes below for some special considerations to keep in mind when you start using Exhibit 3.0.

Note About Exhibit 3.0 Staged (beta)

Exhibit 3.0 Staged allows for much larger data sets but does not support all the features from Exhibit 2 or even all the features from Exhibit 3.0 Scripted mode. See the list of differences in functionality between Scripted and Staged modes for more details.

New Exhibit 3.0 Release URL

With Exhibit 2, you did not have to install a local set of Exhibit files if you called Exhibit from the simile-widgets.org site. The URL has been updated for Exhibit 3.0 Scripted:
`http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-api.js`.

Alternatively, you can download and set up Exhibit 3.0 Scripted on your Web server instead. See the install instructions.

Validate Your JSON Data First

Exhibit 3.0 uses native browser JSON libraries that implement a much stricter implementation of the JSON specification. A one-off extension to upgrade JSON was added to the selection of Exhibit extensions.

Otherwise, you may need to run your existing Exhibit JSON through JSONlint to re-qualify it for use with Exhibit 3.0

Change Your Exhibit Data Link Format

Change <link rel="exhibit/data"/> to <link rel="exhibit-data"/>.
The former use, with (/), is deprecated and will cease to work in future versions.

Using Babel

Babel is a data translation service. There are some important changes in this release to how Exhibit works with Babel.

Supply a URL: If you use Babel for data translation, you must supply a URL to Babel by appending "babel=<url>" to your exhibit-api.js script tag.

Babel Usage Note (Optional): If you rely on Babel (RDF/XML, N3, Excel, an Exhibit page, KML, JPEG, TSV importers), consider running Babel yourself, or download the transformed data if you don't need to actively transform the original data. Or you may want to consider maintaining the data in a format that doesn't depend on Babel.
Babel may not be provided as a public service in the future.

Using jQuery

If you are using jQuery, you may not need to load it separately.
Exhibit loads jQuery 1.7.1. It will not load it if jQuery is detected.

HTML5

If you are using HTML5 with Exhibit, you should update your configuration language to the HTML5-compatible version in order to maintain valid HTML5.

The Exhibit attribute-based configuration has changed for HTML5. A compatibility mode remains for Exhibits in XHTML files. HTML5 does not support XML namespaces, providing a new custom attribute mode in its stead. Moving from Exhibit 2.2.0 in XHTML to Exhibit 3.0 in HTML5 requires changing all attributes prefixed with ex: to be prefixed with data-ex- instead.

In addition, all capital letters within the attribute name should be converted to a hyphen and lower case, e.g., `ex:itemTypes` becomes `data-ex-item-types`. The HTML5 data attribute API treats capitalization differently during document processing and when attribute access occurs, necessitating the change to hyphenation.

See the Authoring section for more info on using Exhibit with HTML5.

Views: Toolbox Parameter

In Exhibit 3.0, the semantics of the toolbox parameter for views (ex:showToolbox, or data-ex-show-toolbox for HTML5) have been unified. All views now have ex:showToolbox, a boolean, set to true by default. The toolbox is displayed by default, so you don't need to hover with the mouse to reveal it.

A new parameter (ex:toolboxHoverReveal), a boolean, set to false by default, will restore the old behavior of hovering to see the toolbox. See the section on Views for more information.

New Control Panel

A new component, the control panel, was added to Exhibit 3.0 to contain Exhibit-wide widgets, like the new Bookmarking widget.

History File

You no longer need a `__history__.html` file to accompany your Exhibit files.

Your Exhibit Code Customizations

If you wrote your own code to extend, augment, or supplant Exhibit 2 code, you almost certainly will have to re-examine how it operates for Exhibit 3.0.
See the developer documentation for more information on customizing and extending Exhibit 3.0.

Feature Map for Exhibit 3.0 Scripted

Examine these lists of Exhibit features to make sure the features you currently use in Exhibit 2.x have been adapted for Exhibit 3.0. This list will change over time, as more features are rewritten for Exhibit 3.0, so check back often.

Views

Some Exhibit 2.x views are not (yet) available in Exhibit 3.0. This table shows which Views are implemented in Exhibit 3.0.

Exhibit 2.2.0 feature	Exhibit 3.0 status
Tile	Fully implemented
Tabular	Fully implemented
Thumbnail	Fully implemented
Timeline Extension	Fully implemented
Map Extension	In progress
Chart Extension	Not yet implemented
Timeplot Extension	Not yet implemented
Calendar Extension	Not yet implemented
Editing	Hooks provided, view not yet implemented

See the Views documentation for more information on Exhibit views.

Facets

Some Facets available in Exhibit 2.x are not (yet) implemented in Exhibit 3.0.

Exhibit 2.2.0 feature	Exhibit 3.0 status
List	Fully implemented
Cloud	Fully implemented
Text Search	Fully implemented

Numeric Range	Fully implemented
Alphabetic Range	Fully implemented
Hierarchical	Fully implemented
Slider	See below[1]
Image	No plans to implement

(1) The Slider Facet can probably be better implemented using new elements available in HTML5. The pre-HTML5 version may be made available as a backwards compatibility headed towards deprecation. This will not take place during the initial phases of Exhibit 3.0 development.

See the Facets documentation for information on sorting, filtering, and searching data with facets.

Exporters

Exhibit 2.2.0 feature	**Exhibit 3.0 status**
Exhibit JSON	Fully implemented
RDF/XML	Fully implemented
Semantic MediaWiki	Fully implemented
Tab-Separated Values	Fully implemented
BibTeX	Fully implemented
Facet Selection URL	Made obsolete by new bookmarking system

Importers

Exhibit 2.2.0 feature	Exhibit 3.0 status
Exhibit JSON	Fully implemented
Google Spreadsheet	Fully implemented
Generic JSONP Framework	Fully implemented
Babel-based[1]	Fully implemented[2]
Generic JSON Framework	No plans to implement
Generic XML Framework	No plans to implement
Generic HTML Table Framework	No plans to implement[3]
RDFa	No plans to implement[4]

(1) Babel-based importers include BibTeX, Excel spreadsheet, Exhibit JSON, Exhibit page, JPEG, N3, RDF/XML, Tab-Separated Values.

(2) While Babel-based importers are fully implemented, users must now supply a Babel installation URL to their Exhibit in order to take advantage of Babel's import Web service. Note that the public Babel services formerly provided by previous SIMILE project member organizations may be turned off in the future.

(3) The Exhibit 2.2.0 Generic HTML Table Framework for importing relied on an XPath content extractor built into Babel and faces the same Babel caveats for running it successfully.

(4) The RDFa importer loaded an externally-hosted script from within itself. This qualifies it for being an extension instead.

Widgets

Exhibit 2.2.0 feature	Exhibit 3.0 status
Toolbox	Fully implemented
Exhibit Logo	Fully implemented
Resizable Element	Fully implemented
Options	Fully implemented
Collection Summary	Fully implemented
Legend	Fully implemented
Legend Gradient	Removed[1]
Bookmark	New to Exhibit 3.0
History Reset Development Tool	New to Exhibit 3.0

1) The Legend Gradient widget does not appear to have been in reasonable working order in Exhibit 2.2.0 and was removed.

Localization

The localization system has changed significantly. All locales require a new key-value set to work. While all localizations have been converted to the new system, some new keys were introduced that could use some translation, and some of the old keys were never properly translated before. While those cases will still display in English, please let us know if you can help provide a translation.

Exhibit 2.2.0 locale	Exhibit 3.0 status
English (en)	Fully implemented
French (fr)	Fully converted
German (de)	Fully converted
Spanish (es)	Fully converted
Dutch (nl)	Fully converted
Norwegian (no)	Fully converted
Swedish (sv)	Fully converted

Installing and Setting Up Exhibit 3.0

This section includes instructions on installing and setting up **Exhibit 3.0** for both **Scripted** and **Staged** modes.

See the section *Choosing Between Exhibit 3.0 Scripted and Staged* if you need help deciding which version is right for you.

Exhibit 3.0 Scripted: Installation and Setup

Exhibit Authors or **Content Publishers**: You do not need to install anything. Simply include the scripts hosted at simile-widgets.org in your page:

```
<script src="http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-api.js"
type="text/javascript"></script>
```

Exhibit Developers: Download and run Exhibit locally on your web server as described below.

Installing Exhibit 3.0 Scripted

[Check the developer documentation wiki for up-to-date installation instructions: https://github.com/zepheira/exhibit3/wiki/Installation]

Developers interested in working on Exhibit 3.0 Scripted can follow these directions to get their environment set up.

Exhibit authors do not need to install the Exhibit developer environment. See the note above about running Exhibit against a central Exhibit server.

Requirements

- Any HTTP server
- A standard Web browser with Web development mode tools (Internet Explorer, Google Chrome, Mozilla Firefox, Safari)
- Git
- GitHub account (optional)
- Java (optional)
- Ant (optional)
- JSCoverage (optional)

Depending on your operating system, you may already have an HTTP server at your disposal. Mac OS X and many variants of Linux include one. If you are unfamiliar with how to acquire and use one, development on Exhibit may not be for you.

Acquiring the Code

If you're planning to contribute your work back to the Exhibit open source project, consider getting a GitHub account so you can fork the project and issue pull requests later to fold your changes back into the main branch.

You can get the code by viewing the project front page and using your preferred method of Git access to clone the main repository. For example, for read only access:

```
% git clone git://github.com/zepheira/exhibit3.git
```

Serving Code

From within the repository, the code that needs to be served can be found at scripted/src/. Making the entire scripted/src/ directory available from your HTTP server is sufficient. Access your deployed Exhibit in your pages by using:

```
<script src="http://yourserver/scripted/src/exhibit-
api.js?bundle=false"></script>
```

Using the `bundle=false` parameter is highly recommended, unless you are testing the bundled mode of operation. Use shift+reload to make sure you're loading the latest scripts in your browser.

Making Changes

As you make changes to the code, make sure to test how your changes affect the project as a whole. Use

```
% ant test
```

from the command line to run the existing unit test suite as well as lint for JavaScript code style checking. To test just one module, use:

```
% ant -Dmodules='[space separated list]' qunit
```

Unit tests do not cover user interaction. The project doesn't have a solution for integration tests right now. Check the Exhibit demos or check your own work across browsers to make sure your changes haven't had a negative impact on the full experience.

All commits are run through a continuous integrator, which will alert the development team to any problems with unit tests on the main branch.

Documentation

We use JSDocToolkit in-code documentation to produce some of our API documents.

More Tests

The Exhibit project can always use more tests. Using JSCoverage, you can see how much of the code is actually tested by our test suite and find precise lines of code that still need some testing to ascertain and maintain correctness. With JSCoverage installed, you can run this code to generate a coverage report:

```
% ant coverage
```

Next Steps

Now that your environment is set up, you should read the following sections of developer doc for Exhibit 3.0 Scripted mode on the wiki at GitHub:

- Component overview of the way Exhibit fits together
- Generated API documentation (Javadoc)
- Reader-oriented API doc for developers working in Exhibit 3 Scripted mode
- Event API doc for Scripted mode

Installing Exhibit 3.0 Staged

[Check the developer doc wiki for up-to-date installation instructions: https://github.com/zepheira/backstage/wiki/Building-backstage]

Exhibit 3.0 Staged runs on the Backstage server. Working with Backstage is not as simple as working with Exhibit Scripted. In addition to HTML and publishing Web pages, Backstage is Java software that acts as a server.

If you're not familiar with Java development, this may not be the project for you to start that learning curve. If you're a content publisher and want to run the server, consider getting some help from a developer to set up the server software, or locate a hosting service to install and manage your exhibits.

Building Backstage

Requirements

You will need all of the following installed on your development system.

- Installing Git
- Installing SVN
- Installing Java 6
- Installing Maven 2
- Installing Ant

Exhibit Installation

Get the source:

```
$ mkdir ~/e3src && cd ~/e3src
$ git clone git://github.com/zepheira/babel.git
$ git clone git://github.com/zepheira/backstage.git
$ git clone git://github.com/zepheira/exhibit3.git
$ svn checkout http://simile-butterfly.googlecode.com/svn/trunk/ butterfly
```

Run the build commands:

```
$ cd babel && mvn install
$ cd ../butterfly && mvn install && ant build
$ cd ../exhibit3/scripted && ant dist
$ cd ../../backstage && mvn package
```

Connect the Exhibit 3 scripts to the Web server:

```
$ cd ../backstage && ln -s ~/e3src/exhibit3/scripted/dist modules/exhibit/api
```

Configuration Procedures

Backstage

The startup script exists at `~/e3src/backstage/backstage` and contains several configuration settings:

- MXMEM specifies the maximum Java heap size. The correct value here depends on several factors including the number and size of data sets, the number of facets in the hosted exhibits, the choice of disk- or memory-based database, as well as uniqueness qualities and size of the faceted properties. Start with the default setting of 1024M and increase it as required, when you run into a Java OutOfMemoryError.
  Note: For the Backstage demos, we observed that each new 100K item data set and two-facet exhibit using a memory-based database, required an additional 1.5G of heap.
- set HOST and PORT to your publicly accessible host and port information. Backstage uses this information to determine your mount point (which can be overridden using the SERVER_ROOT variable), which is in turn used as the base URL for any uploaded data sets.
- set DATABASE_DIR to the directory where the databases corresponding to the uploaded data sets are to be stored. A non-absolute path is interpreted relative to the Butterfly home, and the default is simply "databases" (or `~/e3src/butterfly/databases`)

Web Server

Backstage is developed as a Butterfly application and deployed in a Jetty servlet container which can be configured via its `web.xml` file, `~/e3src/butterfly/main/webapp/WEB-INF/web.xml`.

One setting of interest for Backstage is the session timeout. The lifetime of a Backstage session, which contains in-memory database, is also determined by this value. If the data export feature isn't used, the in-memory database is removed from memory after the session times out.

The following addition to web-app root of the web.xml document specifies a session timeout of 1440 minutes (1 day):

```
<session-config>
    <session-timeout>1440</session-timeout>
</session-config>
```

Execution

Once configured, run:

```
$ ./backstage
```

Authoring Exhibits in Staged Mode

Now you've set up Backstage. Your next step is to author the HTML pages that publish your exhibit. Note that the Backstage server **is not intended** to host your authored HTML. With Staged mode, you will still require your own Web server to host your Exhibit HTML.

See the Exhibit Authoring documentation for details.

Next Steps

Developers running Exhibit 3.0 Staged mode should read the documentation at GitHub specific to Staged mode:

- Exhibit Staged Mode: Authoring
- Backstage Documentation
- HTTP Interface
- Comparing Staged Exhibit to Scripted

Additional Resources

Here are some additional resources from an Exhibit user. They may be helpful if you're installing Exhibit for the first time:

- http://progit.org/book/ch1-3.html
- http://www.jonathansewell.co.uk/index.php/2011/01/06/setting-up-git-in-windows-7/ (Windows)
- http://lostechies.com/jasonmeridth/2009/06/01/git-for-windows-developers-git-series-part-1/ (Windows)
- Installing msysgit for Windows: http://code.google.com/p/msysgit/
- Information on setting up Git (Windows): http://help.github.com/win-set-up-git/
- http://mac.github.com/ (Mac)

Getting Started with Exhibit 3.0 Scripted

Exhibit publishes your data collections on the Web in rich, interactive pages. Exhibit 3.0 has two modes: Scripted mode for running smaller exhibits and Staged mode for larger, server-based exhibits.

This tutorial walks you through the basic steps for creating a simple exhibit in Scripted mode. This example uses XHTML. To see Exhibit 3.0 running with HTML5, see the demo server.

For details on running Staged mode, see the developer documentation on GitHub.

The Basics
Publishing data with the Scripted mode of Exhibit is very simple. You need three basic components.

- **Connection to Exhibit scripts** either remotely or on your own Web server
- **Data file** containing your data, in JSON format (more on that later)
- **HTML** page that calls the Exhibit scripts and displays your page content when loaded in the browser

Tutorial: Creating an Exhibit with Scripted Mode

Let's dive right in. First, you'll build the HTML and data files Exhibit uses to publish data. After you've published an Exhibit, you can refine how the page displays, see which display options work best for you, and tweak the page design as needed.

We assume you know some basic HTML. We'll use sample code and data about MIT's Nobel Prize winners, but you can also substitute your own data. If you get lost or need help along the way, see the finished HTML file at the end of the tutorial.

Creating Your HTML Page
Start with a basic text editor (Notepad on Windows or TextEdit on the Mac).

- Create an HTML file in your text editor.
- If you're following our demo example, copy this text and name the file nobelists.html.

  http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-api.js

Sample Exhibit HTML Text

```
<html>
 <head>
<title> Exhibit | Examples | MIT Nobel Prize Winners</title>

<link href="nobelists.js" type="application/json" rel="exhibit-data" />
    <!-- Replace the URL here with your Exhibit 3.0 script location -->

    <script src="http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-api.js"
type="text/javascript"></script>
    <style>
    </style>
 </head>
<body>
    <div id="main-content">
        <div id="title-panel">
            <h1>63 MIT-related Nobel Prize Winners</h1>
        </div>

        <div id="top-panels">
            <table width="100%"><tr>
             Facets for sorting and browsing go here</tr></table>
        </div>

            <div ex:role="viewPanel" style="padding: 1em 0.5in;">

            <div ex:role="view"
                ex:label="Details"
                ex:viewClass="Tile"
                ex:showAll="true"
               </div>

            </td>
        </tr>
    </table>
</table>

 </body>
 </html>
```

In highlighted code:

(1) Yellow points to your data file, named nobelists.js, identifies it as JSON format, and declares that it contains the data for this exhibit.

(2) Blue describes where to find the Exhibit JavaScript needed to run Exhibit: http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-api.js

(3) Green shows the Exhibit codes that publish a basic view of your data.

This is the most basic HTML text for an Exhibit file. We'll start with the bare bones and add browsing controls and styling as we go.

- Save the file as nobelists.html in any folder you choose.

Next, you prepare a data file, in a format Exhibit can read (JSON), containing your data collection, and save it to the same folder.

Preparing Your Data File in JSON Format
Your data must be structured and formatted so Exhibit can read, publish, filter, sort, and display the data.

We've prepared a sample data file with data about Nobel Prize winners.

Download the sample data file nobelists.js from the demo server at
http://databench.zepheira.com/demos/nobelists/nobelists.js

It will also be available from the Exhibit 3 wiki page:
http://simile-widgets.org/wiki/Exhibit3

Save nobelists.js to the same folder where you saved the HTML file, nobelists.html.

Exhibit looks for your data to be structured in JSON format.

```
"items" : [
        {   "type":                "Nobelist",
            "label":               "Burton Richter",
            "discipline":          "Physics",
            ""shared":             "yes",
            "last-name":           "Richter",
            "nobel-year":          "1976",
            "relationship":        "alumni",
            "co-winner":           "Samuel C.C. Ting",
            "relationship-detail": "MIT S.B. 1952, Ph.D. 1956",
            "imageURL":
"http://nobelprize.org/nobel_prizes/physics/laureates/1976/richter_thumb.jpg"
        },
```

In this sample JSON-formatted data, each Nobel winner's data is an **item**, containing a series of fields such as **type**, **label** (or name), **discipline**, and so on.

Publishing an Exhibit View
Once you've got the HTML and JSON data files, publishing an Exhibit is quite simple.

- Go to the directory where you saved the two files: `nobelists.html` and `nobelists.js` (using Explorer on Windows or Finder on Mac).
- Drag `nobelists.html` into your Web browser.

Behind the scenes, Exhibit marries your structured JSON data with the HTML display template.

- Notice the page displays each Nobelist's data as listed in the JSON file.

You've published your first Exhibit! You should now see a Web page that shows 63 people's names and information. So far, it's a pretty basic display, using the Exhibit's most basic display, the List View.

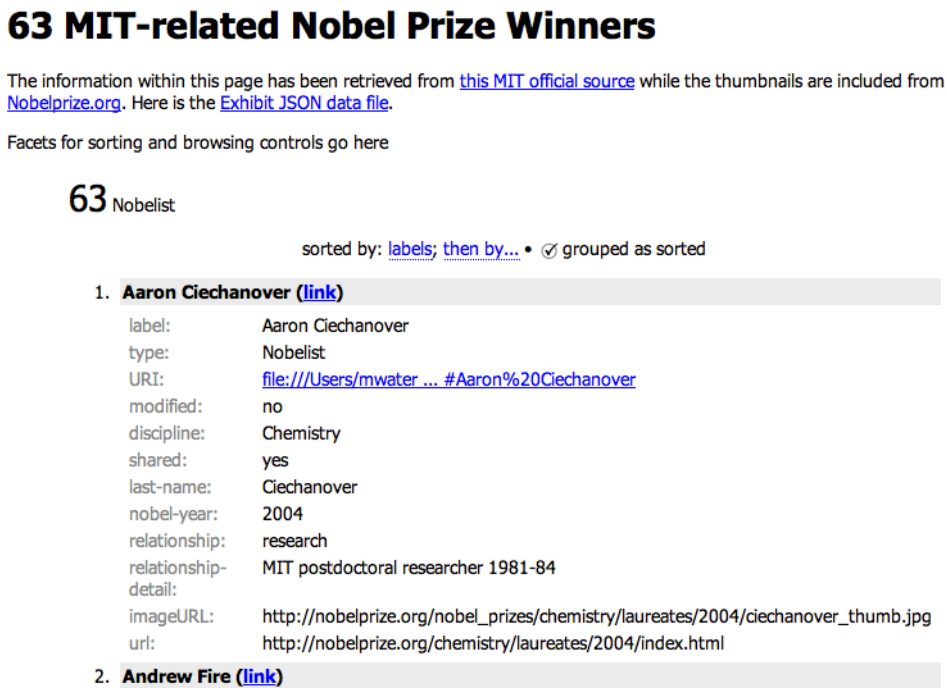


Exhibit makes it easy to add sorting, filtering, and faceting to your data display.
In the next section, you'll learn different ways to present and sort your data.

You'll also have the chance to adjust the page content. Note the title that says "63 Nobelist" instead of "63 Nobelists"—you'll fix that later, too.

Facets: Adding Data Filtering, Searching, and Sorting

Your exhibit so far shows a lot of details but in a long list. Next you'll learn how to add optional **facets** for filtering or sorting data – all done in the browser without complicated programming.

Filtering

Looking at each person's entry in the list view, you see several fields such as "discipline" (the field in which they won their Nobel prize), "relationship" (how they're related to MIT), "shared" (whether they received their prize alone or shared it with others), etc. These fields provide useful ways to sort and filter the data set.

In the HTML code (nobelists.html) find the text "browsing controls here..." and replace it with

```
<div id="top-panels">
            <table width="100%"><tr>
                <td><div ex:role="facet" ex:expression=".discipline"
ex:facetLabel="Discipline"></div></td>
                <td><div ex:role="facet" ex:expression=".relationship"
ex:facetLabel="Relationship"></div></td>
                <td><div ex:role="facet" ex:expression=".shared"
ex:facetLabel="Shared?"></div></td>
                <td><div ex:role="facet" ex:expression=".deceased"
ex:facetLabel="Deceased?"></div></td>
            </tr></table>
        </div>
```

Save the file in your text editor, switch back to your browser, and reload the `nobelists.html` file. Now you should see a row of boxes across the top labeled **Discipline**, **Relationship**, **Shared**, and **Deceased**. These boxes are called *facets*—different angles or aspects of the data that help users analyze the data set.

Notice the gray numbers in front of the values inside these facets. In the facet called Discipline, the number "2" in front of "Peace" tells you there are 2 people who received the Nobel Peace prize. Click on "Peace" and the page now shows only those 2 people (rather than 63 people originally).

By adding those HTML `div` elements as shown above, you've implemented filtering in about 10 seconds. (If you've ever done server-side programming, you might consider how long it would have taken you to implement filtering using whatever server-side publishing frameworks you're familiar with.)

You can also add text search to your exhibit by adding a facet of type `TextSearch`:

```
<div ex:role="facet" ex:facetClass="TextSearch"></div>
```

As you type into the search text box, the exhibit gets filtered.

Sorting

Using Exhibit's basic view, you can already change the sorting order of the Nobelists by clicking on "sorted by" and choosing "labels". But if you want to set a specific sorting order at the beginning – for example, to sort by the discipline and year each person won the prize – you can change `<div ex:role="view"></div>` to:

```
<div ex:role="view" ex:orders=".discipline, .nobel-year"></div>
```

You can also restrict the properties by which your users can sort by using the `ex:possibleOrders` attribute:

```
<div ex:role="view"
      ex:label="Details"
      ex:viewClass="Tile"
      ex:showAll="true"
      ex:orders=".discipline, .nobel-year"
      ex:possibleOrders=".label, .last-name, .discipline, .relationship,
.shared, .deceased, .nobel-year">
</div>
```

Inspecting Your Data File

So far, you've prepared only two files: an .html file and a .js file. One specifies the presentation of the Web page in HTML and the other stores the data in JSON format.

Open up the file `nobelists.js` in your text editor and take a look at it. It's written in the JSON format. There is an array of *items* (think of them as database records). Each item looks like this:

```
{   "type":                  "Nobelist",
            "label":         "Horst L. St\u00F6rmer",
            "discipline":    "Physics",
            "shared":        "yes",
            "last-name":     "St\u00F6rmer",
            "nobel-year":    "1998",
            "relationship":  "research",
            "co-winner":      [
                "Robert B. Laughlin",
                "Daniel C. Tsui"
            ],
            "relationship-detail": "MIT researcher at MIT Magnet Lab",
            "imageURL":
"http://nobelprize.org/nobel_prizes/physics/laureates/1998/stormer_thumb.jpg"
        },
```

That's basically a set of properties and property values. For example, the property shared has the value "yes". Horst L. Störmer won his Nobel prizes with two other people, and so the property co-winner has two values, which are encoded as elements of an array. That just means you list the values between [and], separated by commas.

Properties names must be surrounded by double or single quotes.

Note: Exhibit 3.0 is stricter about JSON formatting than prior versions of Exhibit. If you're seeing errors in JSON data, try using a validator like JSONlint to check your data formatting: http://jsonlint.com/

Required Data Values

Each item in your data file must have a `label` property value. (There are exceptions but this is required for 99% of the cases.) The `type` property value specifies the type of the item; otherwise, the item's type defaults to `Item`. In our data file, we set the type property to

Nobelist.

Aside from these few restrictions, you can put pretty much anything into your data file.

How Does Exhibit Work?

The file `nobelists.html` has a reference to the JavaScript file `exhibit-api.js`. For Exhibit 3.0 Scripted mode, that's how you include Exhibit. Here's roughly what happens when the web page is loaded:

- The code of `exhibit-api.js` is loaded and it automatically pulls in some more code.
- A lightweight database is created and the data files are loaded into it.
- An Exhibit object is created. It reads the various `ex:` attributes in the HTML code to configure its user interface. It then reads data out of the lightweight database and constructs its user interface

For Exhibit 3.0 Scripted mode, remember that everything happens inside the Web browser. The user's Web browser loads the entire data set in memory and performs all computation (sorting, filtering, etc.) locally, which provides much of the power of the Exhibit approach.

Note that so far in this tutorial, you haven't had to install, configure, or manage a database or to write a single line of server-side script (in PHP, ASP, JSP, CGI, etc.).

Learn More About Staged Exhibit

For Exhibit 3.0 Staged mode, a server handles filtering, sorting, and data manipulation. See the Exhibit 3.0 Staged documentation for details.

Lenses: Styling How Your Exhibit Displays

Right now the information about each Nobelist is displayed in a table showing an item's property and value. Exhibit uses *lenses* to style the HTML displaying your data in the browser. For example, you might use lenses to show images for each item in an exhibit.

Insert this `table` inside the `div` element with the id `exhibit-view-panel` (either before or after the inner `div` element that is already there):

```
<table ex:role="lens" class="nobelist" style="display: none;"><tr>
      <td><img ex:src-content=".imageURL" /></td>
       <td>
      <div ex:content=".label" class="name"></div>
      <div>
      <span ex:content=".discipline" class="discipline"></span>,
      <i ex:content=".nobel-year"></i>
      </div>
<div ex:if-exists=".co-winner" class="co-winners">
Co-winners: <span ex:content=".co-winner"></span>
</div>
<div ex:content=".relationship-detail" class="relationship"></div>
</td>
</tr></table>
```

Then, find the `style` element at the start of the HTML code and change it as follows:

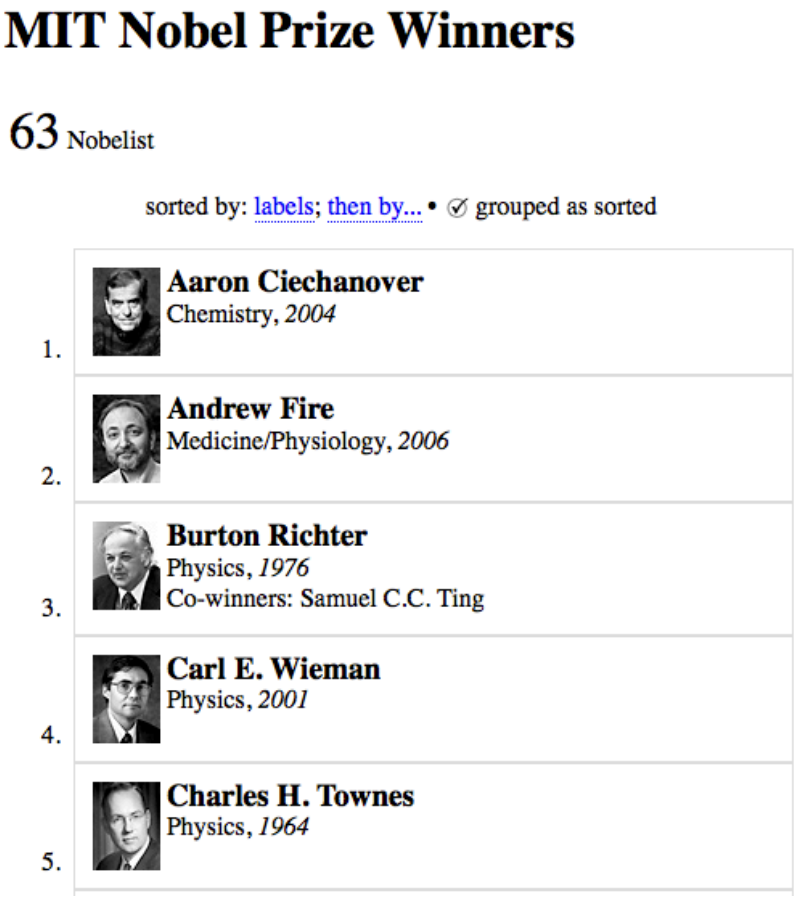
```
<style>
        body {
            font-size: 75%;
            margin: 0;
            padding: 0;
            font-family: "Lucida Grande","Tahoma","Helvetica","Arial",sans-
serif;
            color: #222;
        }
        table, tr, td {
            font-size: inherit;
        }
        tr, td {
            vertical-align: top;
        }
        img, a img {
            border: none;
        }
        #main-content { background: white; }
        #title-panel { padding: 0.25in 0.5in; }
        #top-panels {
            padding:        0.5em 0.5in;
            border-top:     1px solid #BCB79E;
            border-bottom:  1px solid #BCB79E;
            background:     #FBF4D3;
        }

        .exhibit-tileView-body { list-style: none; }
        .exhibit-collectionView-group-count { display: none; }
```

```
        table.nobelist {
            border:     1px solid #ddd;
            padding:    0.5em;
            margin:     0.5em 0;
            display:    block;
        }
        div.name {
            font-weight: bold;
            font-size:   120%;
        }
        .relationship {
            color:  #888;
        }

        div.nobelist-thumbnail {
            float:      left;
            width:      13em;
            height:     10em;
            border:     1px solid #BCB79E;
            background: #F0FFF0;
            padding:    1em;
            margin:     0.5em;
            text-align: center;
        }
        div.nobelist-timeline-lens {
            padding:    1em;
            text-align: center;
        }
    </style>
```

Save and reload in the browser. The Nobelists file looks much better, and you've been able to customize its display to emphasize names and photos for each Nobel winner.



What you have just done is specifying an exhibit *lens template* and some CSS styles. Exhibit uses a lens template to generate the display for each Nobelist. In the lens template, the `ex:content` attribute values specify which item properties Exhibit uses to fill the corresponding HTML elements. For example,

```
<div ex:content=".label" class="name"></div>
```

causes a `div` element to be generated using the label property value of the Nobelist – the person's name.

Adding Images

In addition to generating text, you can also generate images by including an `img` element and specifying its `src` attribute value to be generated, in our example, from the `imageURL` property value in the nobelists.js file:

```
<img ex:src-content=".imageURL" />
```

Customizing Exhibit Attributes

To generate any attribute value in Exhibit, you prepend its name with `ex:` and append `-content`. So if your data file contains a field called `publication-date`, you can display those dates in a lens by using

```
<div ex:content=".publication-date" />
```

Note that some Nobelists have co-winners recorded in the data file, but not all. (We only have information about co-winners who are also related to MIT.) For conditional content like this, you use the `ex:if-exists` attribute value to include a part of the template conditionally:

```
<div ex:if-exists=".co-winner" class="co-winners">Co-winners:
    <span ex:content=".co-winner"></span>
</div>
```

Without the code `ex:if-exists`, the page would show a lot of "Co-winners:" with nothing in the data field.

Schema

Properties

In your Web browser, filter the "discipline" facet to show only "Physics", "relationship" to only "research", and "shared" to "yes". You should get 5 Nobelists.

Note that the Nobelist, Horst L. Störmer, shared his prize with two others: Robert B. Laughlin and Daniel C. Tsui. You can readily see Daniel C. Tsui listed with Störmer, but Laughlin's data doesn't display because he's not a researcher and has been filtered out.

However, Laughlin does appear in the rendition of Störmer, so it would be nice to be able to click on Laughlin's name and get some information about him right there and then. We can do this by adding some schema to our data, to say that co-winner property values, such as "Robert B. Laughlin", actually mean items.

Open the file `nobelists.js` and modify the beginning so that it looks like this:

```
{
    "properties": {
        "co-winner" : {
            "valueType": "item"
        }
    },
    "items" : [
        {   "type":       "Nobelist",
            "label":      "Burton Richter",
            "discipline": "Physics",
            "shared":     "yes",


...the rest of the file...
```

Now, open up another browser window (or tab) and drag the file nobelists.js into it and make sure that the latest version is displayed. If not, hit the browser's Refresh button. Once that's done, switch to the browser window displaying that HTML file and refresh it. Select the same filters as above. You should now see that Robert B. Laughlin's name appears in blue and is underlined. Click on it.

Types

Notice that the number of Nobelists shown on the page is ungrammatical – it reads "5 Nobelist" instead of "5 Nobelists". Modify the beginning of the file nobelists.js as follows:

```
{
    "types": {
        "Nobelist" : {
            "pluralLabel": "Nobelists"
        }
    },
    "properties": {
        "co-winner": {
            "valueType": "item"
        }
    },

   ...the rest of the file...
```

Save your data file, switch to the browser window displaying the .js file, and refresh it. Then switch to the window displaying the .html file and refresh it to see the change.

The key point of this step is that the Web page can look and behave better not just by fine-tuning the HTML code but also by improving the data.

Thumbnail View

Exhibit offers several different Views to display data. With a series of images and short descriptors, the Thumbnail View comes in handy for browsing and sorting a data set like ours.

An exhibit can use several different views. Add this code to nobelists.html after the Lens table:

```
<div ex:role="view"
                ex:viewClass="Thumbnail"
                ex:showAll="true"
                ex:orders=".discipline"
                ex:possibleOrders=".label, .last-name, .discipline,
.relationship, .shared, .deceased, .nobel-year">
                <div ex:role="lens" class="nobelist-thumbnail" style="display:
none;">
```

```
                    <img ex:src-content=".imageURL" />
                    <div><span ex:content=".label"></span></div>
                    <div>
                        <span ex:content=".discipline"
class="discipline"></span>,
                        <span ex:content=".nobel-year" class="year"></span>
                    </div>
                </div>
            </div>
```

Save the file and open it in your browser window. The Thumbnail View is quite handy for showing images and sorting by a particular data field, such as discipline or year.

Timeline View
For data that relates to history or includes time fields, the Timeline Widget adds an interesting dimension to your exhibit.

The `nobelists.js` data file lists the years when the Nobelists won their prizes, so we can plot each one on a time line. To display timelines in Exhibit you need to include a separate utility, the Timeline widget. The Timeline widget is a bit bulky, so Exhibit doesn't include it by default. You have to include the time extension to Exhibit. Open the file nobelists.html, find the reference `to exhibit-api.js` and add the following script element after it:

```
 <script src="http://api.simile-widgets.org/exhibit/3.0.0rc1/extensions/time/time-
extension.js" type="text/javascript"></script>
```

Then add this view after the Thumbnail view code block:

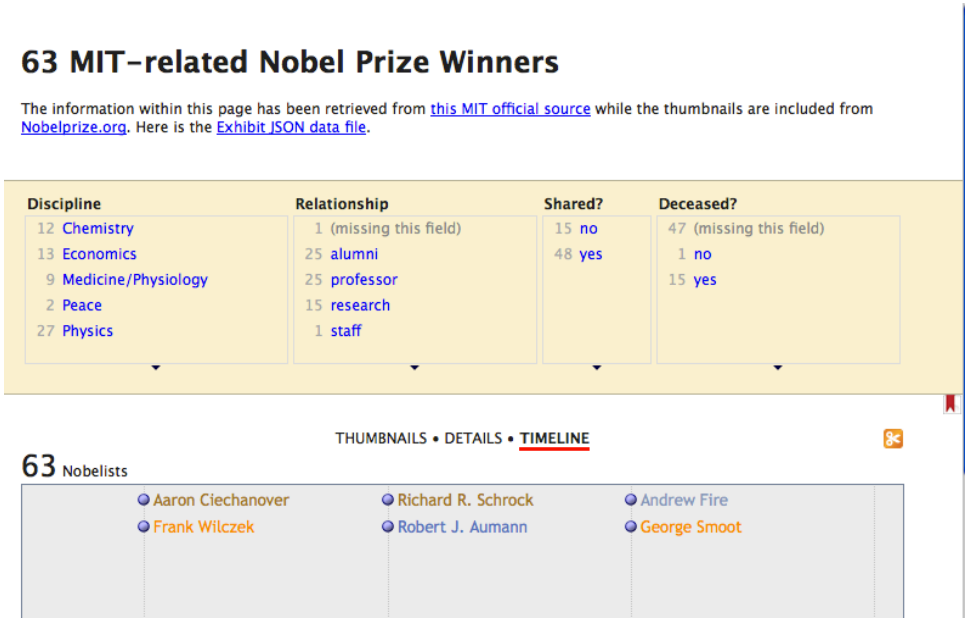
```
<div ex:role="view"
      ex:viewClass="Timeline"
      ex:start=".nobel-year"
      ex:colorKey=".discipline"
      ex:bubbleWidth="150"
      ex:bubbleHeight="150">
<div ex:role="lens" class="nobelist-timeline-lens" style="display: none;">
      <img ex:src-content=".imageURL" />
      <div><span ex:content=".label"></span></div>
      <div>
      <span ex:content=".discipline" class="discipline"></span>,
      <span ex:content=".nobel-year" class="year"></span>
      </div>
</div>
</div>
```

Save and reload in the browser. That's it!

Exhibit Timeline View

This view shows the timeline – notice you can toggle to another view by clicking Thumbnails or Details. Also, note the red flag that appears on the right side of the browsing window, in

the Exhibit control panel – this **Bookmarking** tool lets users copy the current URL to save this view or share it with others.



Formatting Notes

If you want the timeline and tile views to be stacked up, rather than shown as tabs, remove the `ex:role="viewPanel"` attribute to a simple td above the "lens" table.
`<td ex:role="viewPanel">.`

The `ex:start` attribute value in the code you just inserted specifies which property to use as the starting date/time, and the `ex:colorKey` attribute value specifies which property to use to color-code the markers on the time line.

Final HTML Code

Here is the whole HTML file for this tutorial:

```
<html>
  <head>
    <title>SIMILE Widgets | Exhibit | Examples | MIT Nobel Prize
Winners</title>

    <link href="nobelists.js" type="application/json" rel="exhibit-data" />
<script src="http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-
api.js"></script>
    <script src="http://api.simile-
widgets.org/exhibit/3.0.0rc1/extensions/time/time-
extension.js?bundle=false"></script>
    <style>
        body {
```

```
      font-size: 75%;
      margin: 0;
      padding: 0;
      font-family: "Lucida Grande","Tahoma","Helvetica","Arial",sans-serif;
      color: #222;
    }
    table, tr, td {
        font-size: inherit;
    }
    tr, td {
        vertical-align: top;
    }
    img, a img {
        border: none;
    }
    #main-content { background: white; }
    #title-panel { padding: 0.25in 0.5in; }
    #top-panels {
        padding:        0.5em 0.5in;
        border-top:     1px solid #BCB79E;
        border-bottom:  1px solid #BCB79E;
        background:     #FBF4D3;
    }

    .exhibit-tileView-body { list-style: none; }
    .exhibit-collectionView-group-count { display: none; }

    table.nobelist {
        border:     1px solid #ddd;
        padding:    0.5em;
        margin:     0.5em 0;
        display:    block;
    }
    div.name {
        font-weight: bold;
        font-size:   120%;
    }
    .relationship {
        color:  #888;
    }

    div.nobelist-thumbnail {
        float:      left;
        width:      13em;
        height:     10em;
        border:     1px solid #BCB79E;
        background: #F0FFF0;
        padding:    1em;
        margin:     0.5em;
        text-align: center;
    }
    div.nobelist-timeline-lens {
        padding:    1em;
        text-align: center;
    }
</style>
<script type="text/javascript">
```

```
        function deceaseRowStyler(itemID, database, tr, rowIndex) {
            var deceased = database.getObject(itemID, "deceased");
            if (deceased == "yes") {
                tr.style.backgroundColor = "#f88";
            }
        }
    </script>
  </head>
  <body>
    <div id="main-content">
        <div id="title-panel">
          <h1>63 MIT-related Nobel Prize Winners</h1>
          <p>The information within this page has been retrieved from
          <a href="http://web.mit.edu/newsoffice/special/nobels.html"
target="_blank">this MIT official source</a> while the thumbnails are included
from <a href="http://nobelprize.org/" target="_blank">Nobelprize.org</a>. Here
is the <a href="nobelists.js" target="_blank">Exhibit JSON data file</a>.
          </p>
        </div>

        <div id="top-panels">
            <table width="100%"><tr>
                <td><div ex:role="facet" ex:expression=".discipline"
ex:facetLabel="Discipline"></div></td>
                <td><div ex:role="facet" ex:expression=".relationship"
ex:facetLabel="Relationship"></div></td>
                <td><div ex:role="facet" ex:expression=".shared"
ex:facetLabel="Shared?"></div></td>
                <td><div ex:role="facet" ex:expression=".deceased"
ex:facetLabel="Deceased?"></div></td>
            </tr></table>
        </div>
        <div ex:role="viewPanel" style="padding: 1em 0.5in;">
            <table ex:role="lens" class="nobelist" style="display: none;"><tr>
                <td><img ex:src-content=".imageURL" /></td>
                <td>
                    <div ex:content=".label" class="name"></div>
                    <div>
                        <span ex:content=".discipline"
class="discipline"></span>,
                        <i ex:content=".nobel-year"></i>
                    </div>
                    <div ex:if-exists=".co-winner" class="co-winners">
                        Co-winners: <span ex:content=".co-winner"></span>
                    </div>
                    <div ex:content=".relationship-detail"
class="relationship"></div>
                </td>
            </tr></table>

            <div ex:role="view"
                ex:viewClass="Thumbnail"
                ex:showAll="true"
                ex:orders=".discipline"
                ex:possibleOrders=".label, .last-name, .discipline,
.relationship, .shared, .deceased, .nobel-year">
                <div ex:role="lens" class="nobelist-thumbnail" style="display:
```

```
none;">
                    <img ex:src-content=".imageURL" />
                    <div><span ex:content=".label"></span></div>
                    <div>
                        <span ex:content=".discipline"
class="discipline"></span>,
                        <span ex:content=".nobel-year" class="year"></span>
                    </div>
                </div>
            </div>
            <div ex:role="view"
                ex:label="Details"
                ex:viewClass="Tile"
                ex:showAll="true"
                ex:orders=".discipline, .nobel-year"
                ex:possibleOrders=".label, .last-name, .discipline,
.relationship, .shared, .deceased, .nobel-year">
            </div>
            <div ex:role="view"
                ex:viewClass="Timeline"
                ex:start=".nobel-year"
                ex:colorKey=".discipline"
                ex:bubbleWidth="150"
                ex:bubbleHeight="150">
                <div ex:role="lens" class="nobelist-timeline-lens"
style="display: none;">
                    <img ex:src-content=".imageURL" />
                    <div><span ex:content=".label"></span></div>
                    <div>
                        <span ex:content=".discipline"
class="discipline"></span>,
                        <span ex:content=".nobel-year" class="year"></span>
                    </div>
                </div>
            </div>
        </div>
    </div>
  </body>
</html>
```

Summary and Next Steps

This tutorial shows you how to start making an exhibit by creating two files:

- An HTML file with formatting and Exhibit codes, nobelists.html
- Your data formatted in a JSON data file, nobelists.js

Then, by adding more code to the HTML file or the JSON file, you make the exhibit Web page look and behave better—supporting filtering and sorting, providing more views, presenting items in custom ways, etc. Using the Scripted mode of Exhibit, you can accomplish this without ever touching a Web server or a database.

To get more involved with Exhibit, explore the online demos, subscribe to the mailing list, and contribute your expertise to the community.

Exhibit Demos

The Nobelists HTML and JSON files are available on the Exhibit demo server. This example uses XHTML. To see Exhibit 3.0 running with HTML5, see the Senate demo here: http://databench.zepheira.com/demos/senate/html5.html

The Exhibit demos are a great showcase for a variety of ways you can use Exhibit, both Scripted and Staged modes. Use them to get ideas, find code samples, and kick-start your own exhibits.

Developers: Learn More About Exhibit

For software developers who would like to dig into the details of Exhibit coding, see the Exhibit API doc online: https://ci.zepheira.com/job/test_exhibit3/javadoc/index.html.

You can also learn more about customizing and extending Exhibit to add your own widgets and views.

Troubleshooting

If your Exhibit code doesn't display as expected, check these things:

- Does your HTML file call the correct data file in the code?
  ```
  <link href="nobelists.js" […]
  ```
- If you see JSON errors in your browser, double check your data file by running it through JSONlint or another data validator. Exhibit 3.0 applies stricter JSON standards than Exhibit 2 did.
- Is your script tag pointing to a working installation of Exhibit 3.0? The Exhibit community hosts a working installation here:
  `http://api.simile-widgets.org/exhibit/3.0.0rc1/exhibit-api.js`
- If the HTML header and title load, but your data does not display, double check the URL you're using to call the Exhibit JavaScript. Exhibit needs to be loaded and running in your Web server.
- Open your browser's error console to check whether Exhibit reported errors that might help guide you to a solution.

Your Exhibit Data

Publishing an Exhibit calls for an HTML page to display your data plus a data file or location. This section describes how you format and use data with your exhibits. It also describes Exhibit data models and expressions.

What's New in Exhibit 3.0?

There are some notable changes in the way Exhibit handles data in Exhibit 3.0 compared to previous releases.

Stricter JSON Validation

Exhibit 3.0 has stricter standards for JSON data validation. For existing Exhibit 2 users, we suggest you validate your JSON data using JSONLint before publishing exhibits with Exhibit 3.0.

If your data fails to load, you'll need to update it to conform to the JSON specification. You can try to use a one-off extension to upgrade your old JSON for you.

Changes to How Exhibit Works with Babel

The Babel data translation service is no longer integrated in Exhibit. You can still call Babel to translate your data from one format to another, but you need to supply the Babel URL to exhibit.

You will need to supply a URL to Babel by appending "babel=<*url*>" to your exhibit-api.js script tag.

For more information on Babel, see http://service.simile-widgets.org/babel/.

Note about Babel: The Babel service is not guaranteed to run as a public service indefinitely. If you rely on Babel translation services (RDF/XML, N3, Excel, an Exhibit page, KML, JPEG, TSV importers), consider running Babel yourself, downloading the transformed data if you don't need to actively transform the original, or maintaining the original data in a format that does not depend on Babel.

rel Usage Note

For Exhibits published using Exhibit 3.0, change `<link rel="exhibit/data"/>` to `<link rel="exhibit-data"/>`. The former use is deprecated and will not work at a future date.

Usage with HTML5

The Exhibit attribute-based configuration has changed for HTML5. A compatibility mode remains for Exhibits in XHTML files. HTML5 does not support XML namespaces, providing a new custom attribute mode in its stead.

Moving from Exhibit 2.2.0 in XHTML to Exhibit 3.0 in HTML5 requires changing all attributes prefixed with `ex:` to be prefixed with `data-ex-` instead. In addition, all capital letters within the attribute name should be converted to a hyphen and lower case, e.g., `ex:itemTypes` becomes `data-ex-item-types`.

The HTML5 data attribute API treats capitalization differently during document processing and when attribute access occurs, necessitating the change to hyphenation.

Data Export for Scripted Mode: Toolbox UI Element

Exhibit 3.0 Scripted modifies the toolbox UI element behavior. Instead of disappearing and re-appearing based on mouse hovering over a view, the toolbox ("scissors") are by default always visible. The former behavior can be reintroduced with a new configuration setting.

HTTP Input and Output (Staged Mode)

The Backstage server for Exhibit 3.0 Staged mode publishes an HTTP+JSON interface to all its data and functionality. See the HTTP Interface documentation on GitHub for more information.

With Staged mode, you can invoke HTTP Get on the data link URL to export the entire dataset in HTML+RDFa format (not the original format), to facilitate search engine indexing.

See the Authoring documentation for Backstage for details on using the data input URL for Staged exhibits, as well as data export features to help make Exhibit data findable by search engines.

Importing Data Into Your Exhibit

Exhibit 3.0 Staged Mode

Adding data to Exhibit Staged is through HTTP. See the developer documentation about the HTTP Interface for details on data upload, creating a database either in-memory or on disk, and more.

Exhibit 3.0 Scripted Mode

There are two basic ways to add data to a Scripted exhibit:

- Use Exhibit's built-in data importers for data in one of these formats:
  - Exhibit JSON
  - Google spreadsheet
  - Generic JSONP framework
  - Babel-Based importing format

  You need to specify which importer to use, and for Babel, supply the URL of a Babel installation (your own or a centrally available Babel service).
  Babel-based importers include these input formats:

  - BibTeX
  - Excel spreadsheet
  - Exhibit JSON
  - Exhibit page
  - JPEG
  - N3
  - RDF/XML
  - Tab-separated values

- Convert your data to JSON manually, with a Babel service or by hand, and then publish the exhibit calling your JSON file.
  Exhibit 3.0 relies on the full JSON standards, which were not fully enforced in previous versions of Exhibit. Use a JSON validator such as JSONLint to make sure your JSON is formatted properly.

The following sections offer more details on formatting and importing your data into Exhibit 3.0 Scripted mode.

Creating, Importing, and Managing Data

Exhibit's database natively understands data in its own format (a JSON format), but there are a number of ways to use data in other formats. If you have existing data in another format or if you prefer another format, you can

(1) Use the Babel service to convert your data into Exhibit's JSON format » Details
(2) Use an importer to convert your files into Exhibit's JSON format on-the-fly » Details

Manually Creating and Managing Exhibit Data

To create and manage data in files in Exhibit's JSON format, you just need a decent text editor (see the list of recommended tools).
Start by entering this code into your text editor

```
   {
  "items":
 [
        ]
   }
```

Save it in the same directory where your web page (HTML file) is stored. Give it a .js or .json extension (this is optional, done just by convention).

Be careful: note that there are both braces { } and brackets [ ]. Loosely speaking, braces { } are used to wrap many properties of different names, while brackets [ ] are used to wrap several things in a list.

In the code above, your data records, or *items* in Exhibit's terminology, go in between the brackets. Here is the same code with three items:

```
{
        "items": [
            {   "label":              "John Doe",
                "type":               "Person",
                "age":                "36",
                "likes":              "Mary Smith",
                "favorite-color":     ["blue", "yellow"]
            },
            {   "label":              "Mary Smith",
                "type":               "Person",
                "married-to":         "Joe Anderson",
                "job":                "Doctor",
                "worksAt":            "Boston General Hospital",
                "hobby":              ["painting", "karate"]
            },
            {   "label":   "Boston General Hospital",
                "type":    "Place",
                "city":    "Boston"
            }
        ]
    }
```

Notes:

- This example shows two types of items: `Person` and `Place`. You can use as many as you want, and name them however you want. It's your data--you're the boss. Exhibit doesn't require you use a global schema for your data.
- Items of the same type, `Person` in this case, don't have to have the same properties all filled in. So, John has `age` but Mary doesn't. And Mary has `job` while John doesn't. Etc. etc. Fill in whatever information you have. You'll get some value of out Exhibit even with incomplete, messy data.
- The code shown here is neatly formatted and aligned, but it doesn't have to be. You can manage your file in whatever way suits you, so you won't make mistakes. Your data is your business.

Data Formatting Notes

Here are some formatting issues to keep in mind when formatting your data:

- Watch out for { } vs. [ ]. Each item in the code above is wrapped in { } while a list of things like `"blue", "yellow"` is wrapped in [ ].
- Watch out for commas. They are used to separate properties within { } and elements of a list within [ ]. Use commas only where needed. Do not put a comma after the last property in a pair of { }. Browsers can get very picky about misplaced commas.
- Put quotation marks around all property names, e.g., `"job"`, or `"co-author"`.

Your exhibit can include one or more data files. Each data file can contain any number of items (or none at all). It can also contain information about types and properties. You can decide how to split your data among several files.

Converting Data Using Babel

You can use the Babel web service to convert data from various formats into Exhibit's JSON format.

Babel-Based Importers

Babel-based importers for pulling data into Exhibit include BibTeX, Excel spreadsheet, Exhibit JSON, Exhibit page, JPEG, N3, RDF/XML, and tab-separated values, or TSV. While these importers are fully implemented, users must supply a Babel installation URL in their Exhibit code in order to use Babel's import service.

To call Babel you need to supply the Babel service's URL by appending "babel=<*url*>" to your exhibit-api.js script tag.

For more information on Babel, see http://service.simile-widgets.org/babel/.

Note about using Babel: The Babel service is not guaranteed to run as a public service indefinitely. If you rely on Babel translation services (RDF/XML, N3, Excel, an Exhibit page, KML, JPEG, TSV importers), consider running Babel yourself, downloading the transformed data if you don't need to actively transform the original, or maintaining the original data in format that does not depend on Babel.

Babel gives you the option of entering the URLs to your data files, uploading your data files from your computer, and just simply pasting your data into a text box.

At this time, the two most popular formats we support are BibTeX and Tab-Separated Values (TSV). While BibTeX is a special treat for the academically inclined (more details here), TSV is useful for everyone.

If you have data in tab-separated format, you can use Babel to convert the data to JSON and then load it into Exhibit. Exhibit will not convert your data for you.

Converting Data at Load Time

Exhibit comes with a few importers that can either parse other formats themselves or convert the data through Babel at a Babel service URL you provide.

To import other formats, specify the following importer types in your Exhibit code:

Excel files: use any of the following
- application/msexcel
- application/x-msexcel
- application/vnd.ms-excel
- application/x-excel
- application/xls
- application/x-xls

RDF/XML files: application/rdf+xml
N3 files: application/n3
BibTeX: applicationl/x-bibtex

If you can manually convert your data through Babel, you should be able to import it dynamically into your exhibit using this method.

Note that this method slows down your exhibit because your data needs to travel through Babel first. We recommend that you do this only while developing your exhibit. Once your exhibit is finished, convert your data manually through Babel, save the result, and link your exhibit to the converted data instead.

Google Spreadsheet Importer

Refer to How to make an exhibit from data fed directly from a Google Spreadsheet.

JSON Maker (Excel Spreadsheet)

Jon Bogacki has written a macro enabled Excel Spreadsheet to convert Excel spreadsheet data into Exhibit JSON format: JSON Maker for SIMILE Widgets. The spreadsheet provides a simple interface to set your data Types and Properties.

Understanding an Exhibit Database

Each exhibit created with Exhibit 3.0 Scripted mode has a database implemented in JavaScript that stores the exhibit's data and lets other parts of the exhibit query the data they need. This database is different from traditional (relational) databases that you might be familiar with, not only because it is implemented in JavaScript but because its *data model* is different.

1. Data Models

Different *data models* are different *conceptual* ways for describing and dealing with data. For example, if you were to write George Washington's biography, here are three different data models you might use:

> Write his biography as prose in a book, broken down into chapters but essentially organized in a *sequential* manner, intended to be read from start to end.
> Write his biography in several web pages, with links between them, so the reader can travel instantly from one event in Washington's life to another related event no matter how far apart in time those events occurred.
> Write his biography in a table with several columns, including the names of events, the times when they happened, locations where they happened, the people involved or affected, etc., so that the reader can sort, group, and filter the events, or re-visualize them on time lines and maps.

Different data models are suited for different purposes. Prose might be nice to read a child to sleep, or to provide commentary and analysis in addition to face. Tables are great for manipulation of the data, and re-visualization helps to present information in different ways. You don't need to understand data models too deeply. Just know that different data models exist and are designed for different purposes.
Exhibit has its own data model, which consists of *items*, *types*, *properties*, and *property values*.

2. Items

Each Exhibit database contains zero or more *items*. If it helps, you can think of items as *records* in traditional (relational) databases. An item represents something, anything—a person (Peter Pan), an object (the book called "The DaVinci Code"), a concept (beauty), etc. It's up to you to decide what constitutes items in your own exhibit.

Identifiers

Each item has a unique *identifier* (or ID for short) that uniquely identifies the item within the exhibit. So two different items in an exhibit should have two different IDs – just as any two different people in the U.S. should have different social security numbers. If you accidentally assign the same identifier to two items, they will be considered the same by Exhibit.

An identifier is just a string—a short piece of text. There is really no restriction on what text can make up an identifier, but we would recommend something meaningful to you: "DaVinci Code", "Peter Pan", and "Beauty" would make good identifiers.

Although items have identifiers, you don't usually deal with identifiers directly. But we want to mention identifiers first just because we need to talk about them in various places later on.

Labels

In addition to an identifier, each item also has a *label* that is used to textually label the item in many cases when Exhibit needs to show the item in the web page.
Labels don't have to be unique. For example, two items (people) with IDs "John Doe #1" and "John Doe #2" can both have the label "John Doe". In most cases, you can use the same text for both the label and the ID of an item. In fact, Exhibit automatically assigns an item's label as its ID if you don't explicitly provide its ID.

3. Types

Each item also has a *type*. For example, the type of the item identified as "Peter Pan" would be "Person", the type of "The DaVinci Code" would be "Book", the type of "Beauty" would be "Concept".
Once again, Exhibit doesn't place any restriction on what constitutes types in your exhibit. You make that decision for your own data. Remember our motto for Exhibit: Your data, your business!

If you don't explicitly assign the type to an item, Exhibit sets the item's type to "Item".
Just like items, types also have IDs, which are just strings, e.g., "Book", "Beauty", and "Concept". Types also have labels – more on labels later on.

4. Properties and Property Values

Now the fun part begins. Each item can have zero or more *properties* (otherwise known as attributes, fields). For example, the item "Peter Pan" would have

A "gender" property
A "member-of-gang" property

The item "The DaVinci Code" would have

An "author" property
A "number-of-copies-sold" property

The *property value*, or just *value* for short, of the "gender" property of "Peter Pan" is "male", and the value of the "member-of-gang" property of "Peter Pan" is "The Lost Boys". Similarly, the "author" property value of "The DaVinci Code" is "Dan Brown", and the "number-of-copies-sold" value is 6,347,343 or however many copies it was sold.

Value Types

Note that while the "gender" property value mentioned, "male", is text, the "number-of-copies-sold" property value is a number. So, property values can be of different *value types*:

text	`"Hello World!"`
number	`"67.5"`
date	`"2006-12-08"`, see ISO 8601 format, also see Working with dates
boolean	`"true"` or `"false"`
url	`"http://www.google.com/"`
item	More about this soon

All property values of a property (e.g., "number-of-copies-sold") have the same value type ("number"). It is not possible to say that the "number-of-copies-sold" property value of "The DaVinci Code" is 6347343 while the "number-of-copies-sold" property value of "Lord of The Rings" is "so many I can't count" because the first value is a number and the second is text.

Don't forget to use quotes around property values.

Item Value Type

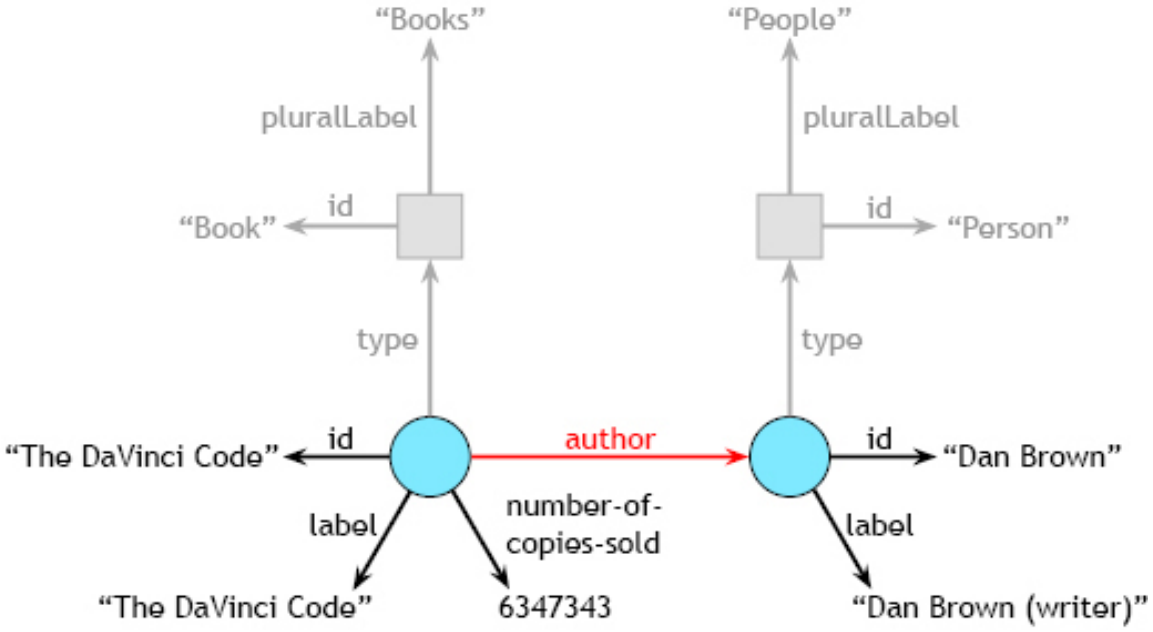
We noted above that the "author" property value of "The DaVinci Code" is "Dan Brown". It's OK to consider that property value to be of value type "text", but since Dan Brown is actually a person, there's more we can do.

We can create another item of type "Person", with ID "Dan Brown", and with label "Dan Brown (writer)". And then, we can declare that "author" property values are of value type "item". When we say the "author" property value of "The DaVinci Code" is "Dan Brown", we actually make a *relationship* between the item "The DaVinci Code" and the item "Dan Brown". The property value "Dan Brown" is no longer just text, but it *identifies* another item.

To see this principle demonstrated, examine at the Getting Started with Exhibit example of MIT Nobel Prize Winners, where the property "co-winner" is changed from a value to an item.

5. Graph-Based Data Model

The relationship between "The DaVinci Code" and "Dan Brown" mentioned previously is shown as a red arrow in this graph representation of the data:



Relationships are properties that link items to items. Other properties link from items to text, numbers, dates, booleans, and URLs. So, the *value type* of a relationship property is "item".

Note that there are two different concepts of *types* here: types of items (e.g., "Book", "Person") and value types of properties (e.g., "number", "date", "item"). When we say that the *value type* of the "author" property is "item", we don't say anything about the types of the authors themselves. Books can be written by individual people, small groups, large organizations, or even a faceless, nameless mob.

Although we say that the item "Dan Brown" is an "author" property value of the item "The DaVinci Code", there should be no implication that somehow the item "Dan Brown" is smaller or less important a thing than the item "The DaVinci Code". We could have also structured the properties such that the item "The DaVinci Code" is a "has-written" property value of the item "Dan Brown" and reversed the red arrow. It doesn't matter to Exhibit which direction you pick for a relationship, so just pick the direction most natural to you yourself.
Ready to learn more? Go on to Learn how to use expressions in your Exhibit.

Exhibit Expressions

Data in an Exhibit database can be represented as a graph, as in this example:

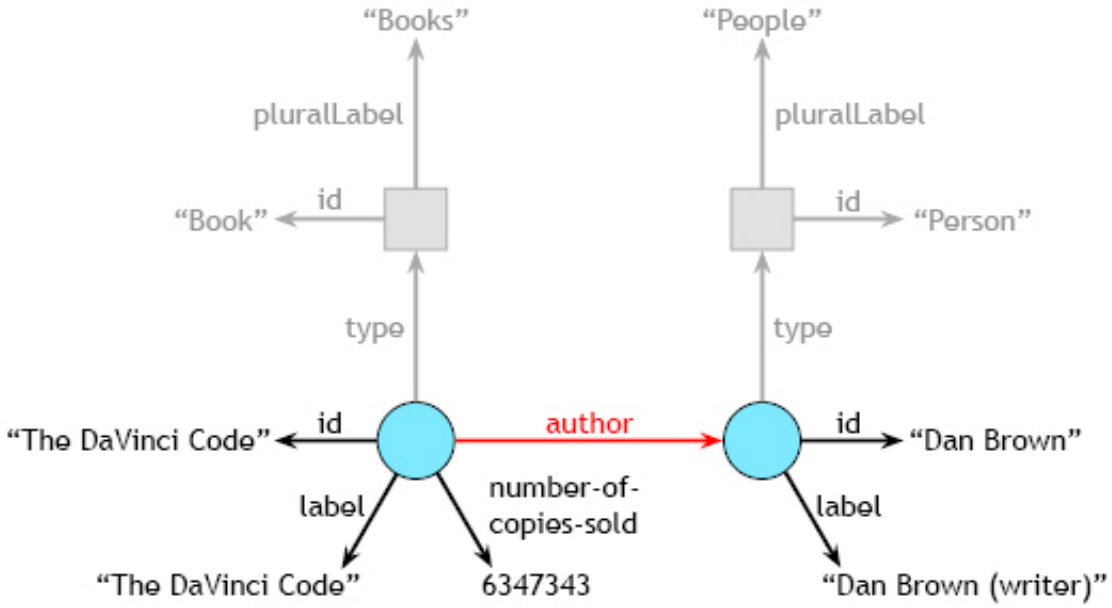


Exhibit expressions are used mainly to move along paths through paths and items, such as the path depicted in the graph. That is, given some nodes in the graph (whether circles/items or arrows/properties), evaluating an Exhibit expression retrieves other nodes (items or properties) that are related.

Exhibit moves along such paths by means of *expressions*. An Exhibit expression consists of a single *path*. A path consists of a sequence of one or more property IDs, each preceded by a *hop operator*. The `.` hop operator traverses along an arrow (forward, or away from an originating circle/item) while the `!` hop operator traverses against an arrow (backward, or toward a circle/item).

For example, given the "The DaVinci Code" item node (the blue circle on the left in the graph above), evaluating `.author.label` returns "Dan Brown (writer)". Given the `6347343` value node, evaluating `!number-of-copies-sold.author` returns the item node "Dan Brown" (that is, you'll get the whole item/object, not just its name). Evaluating `!number-of-copies-sold.author.id` returns the value node "Dan Brown" (the id value, not the item itself).

Here are some more examples. You should be able to imagine for yourself, based on the wording of the properties, how the data might appear in a graph like the one above:

> evaluating `.hasAuthor.teachesAt.locatedIn` on some papers returns the locations of the schools where the authors of those papers teach.
> evaluating `.spouseOf!parentOf` on some people returns their parents-in-law.
> evaluating `!shot!arrested` on John F. Kennedy returns the police officers who arrested his assassin.

A path can also start with one of a few predefined variables, currently including

`value` (referring to the current item or value on which the expression is being evaluated) and

`index` (referring to the index of the current item/value in a sequence of items/values)

`value` is understood if there is no such variable at the beginning of a path. That is, you can also write `.spouseOf!parentOf` as `value.spouseOf!parentOf`.

Authoring Your Exhibits

Authoring an exhibit is as simple as creating an HTML page – you don't need to learn programming and can get started by cutting and pasting existing HTML.

This section offers tips on writing the web pages that publish your Exhibits.
Topics include:

- What's New in Exhibit Authoring?
- What's Changed Between Exhibit 2 and Exhibit 3.0?
- Moving Your Exhibits to Exhibit 3.0
- Authoring Overview
- Using HTML5 with Exhibit
- Reference Documentation, Additional Resources, and Demos

Notes on Authoring Exhibits in Staged Mode
Developers working with Exhibit 3.0 Staged mode should also consult the Authoring documentation on GitHub.

What's New in Exhibit Authoring?

Essentially, the way you write your Exhibit web files remains the same as in previous releases. You create an HTML file that calls the Exhibit scripts and points to your Exhibit JSON data file(s).

But there are a few items you need to note.

Authoring Notes for Exhibit 3.0 Scripted (rc1)

To publish an Exhibit with Scripted mode, you code an HTML exhibit page that publishes your data. See the Previous Release Comparison documentation for information on views, facets, and lenses supported in Scripted mode.

Authoring Notes for Exhibit 3.0 Staged (beta)

Creating an exhibit in Exhibit 3.0 Staged also calls for an HTML page that publishes your exhibit, using data stored in the Backstage server. Note that the Backstage server **doesn't produce or** host your HTML page. As an author, you still need to create and post your HTML on your own web server.
See the developer documentation on GitHub for details on authoring a Backstage-hosted Exhibit.

What's Changed Between Exhibit 2 and Exhibit 3.0?

Most of the codes and procedures have stayed the same for authoring exhibits. But there are some important notes for exhibit authors.

The expression language remains the same.
See the Previous Release Comparison documentation for details.

Authoring in HTML5 does require some changes to the web page code that publishes your exhibit. The Exhibit attribute-based configuration has changed for HTML5. See the HTML5 section below for information on authoring an exhibit with HTML5.

Moving Your Existing Exhibits to Exhibit 3.0

Moving a published exhibit from Exhibit 2 to Exhibit 3.0 is fairly simple.

Note: If you're moving a smaller exhibit in Exhibit 2 to take advantage of the scalability offered by Exhibit 3.0 Staged, you should first upgrade to Exhibit 3.0 Scripted, get used to the new data validation and functionality changes, and then move from Scripted to Staged mode.

Move your Exhibit 2 exhibit to Exhibit 3.0 Scripted:

- In your HTML file, point the Exhibit script source to the location of Exhibit 3.0 JavaScript.
- Check that your data validates correctly. If you encounter errors, validate your JSON files at JSONLint.
- Check which features are implemented in Exhibit 3.0 that are used in your HTML
- View your exhibits in your web browser.

If your dataset is large, you'll also want to scale it up by moving from the Scripted to Staged mode:

- Follow the install and setup instructions for setting up Backstage.

- In your HTML page that publishes the exhibit, point to the data on Backstage. See the HTTP documentation for Exhibit 3.0 Staged to learn more.

  There are significant differences in functionality between the Scripted and Staged modes of Exhibit 3.0.
  For example, Staged mode doesn't support maps views, timelines or search facets.
  See the list of differences between Scripted and Staged mode to check which features are not yet supported in Staged mode.

- Remove or comment-out views and facets that are not supported in Staged mode.
- Load your Exhibit HTML in a browser to see your Staged exhibit.

  Reminder: Backstage does not serve as a Web server. You still need to author an Exhibit HTML page and host it separately on your Web server.

See additional notes on creating a Staged exhibit in the Backstage documentation on GitHub.

Authoring Overview

You create an exhibit by authoring an HTML page that points to the exhibit scripts (Scripted mode) or to your Backstage server (Staged mode).

Authoring an exhibit is fairly straightforward:

- Create an **HTML** page that will display your data.
  Using HTML5? See the coding notes about HTML5 publishing below.
- For **Scripted** mode, specify the location of your Exhibit scripts.
  For example:
  ```
  <script src="http://yourserver/scripted/dist/exhibit-
  api.js?bundle=false"></script>
  ```
  For **Staged** mode, you add a data URL. See the Authoring documentation to learn more.
- Exhibit uses the default **View** to publish your data: Tile View.
  Notice the control panel that appears above the Tile View, with clickable icons for bookmarking this view or exporting your data.
- Add **Facets** to let users sort, search, and filter your data.
- Add **Lenses** to control which parts of your data appear in the exhibit.
- Use **HTML styles** to customize the look and feel of your published exhibit.

See the Getting Started tutorial for detailed, step-by-step instructions on publishing your first exhibit.

To see the variety of ways you can use Exhibit to publish interactive web pages, look through the Exhibit demos on the continuous integration server. View the page source to see how the author uses views, facets for sorting and filtering, and lenses for choosing which data is most important to feature.

Example Code: Staged Mode

The Encyclopedia of Life example publishes an exhibit with over a hundred thousand items using the following page code.

The HTML is fairly brief, establishing first a link to the data on backstage with the `<script src="http://backstage[…]>` tag and then using a lens to customize the view that publishes the content. Facets in the sidebar let user sort results by the year it was classified and who classified it.

Note that the author of an exhibit in Staged mode would load the data on Backstage, and would also place the exhibit HTML file on his/her own Web server to publish the exhibit. Backstage does not create or host the exhibit HTML.

See the Encyclopedia of Life demo here:
http://databench.zepheira.com/eol/

```
<html xml:lang="en" lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ex="http://simile.mit.edu/2006/11/exhibit#">
  <head>
    <title>Encyclopedia of Life</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
    <link href="http://backstage.zepheira.com:8181/backstage/data/mem/eol" rel="exhibit/data"
/>
    <script src="http://backstage.zepheira.com:8181/backstage/exhibit/api/exhibit-
api.js?autoCreate=false&postLoad=true&js=http://backstage.zepheira.com:8181/backstage/api/back
stage-api.js" type="text/javascript"></script>
  </head>

  <body>
    <div id="title-panel">
      <h1>Encyclopedia of Life</h1>
      <p>This <a href="http://simile-widgets.org/exhibit3/">Exhibit 3.0 Staged</a> demo
takes a different look at the <a href="http://eol.org/">Encyclopedia of Life</a>.</p>

    </div>

    <div id="content">

      <div ex:role="lens" class="lens" style="display: none;">
        <a ex:href-content=".link" target="_blank">
         <div ex:content=".commonName" class="label"></div>
          <div ex:content=".name" class="taxon"></div>
          <img ex:src-content=".image" />
        </a>
        <p class="citation">Classified in <span ex:content=".citation"></span></p>
      </div>

      <table width="100%">

        <tbody>
          <tr valign="top">
            <td>
              <div ex:role="view"></div>
            </td>
            <td class="sidebar">
```

```
              <div ex:role="facet"
                ex:expression=".year"
                ex:sortDirection="reverse"
                ex:facetLabel="Year Classified"
                ex:height="20em"
                ></div>
              <div ex:role="facet"
                ex:expression=".classifiedBy"
                ex:facetLabel="Classified By"
                ex:height="20em"
                ></div>
            </td>
          </tr>
        </tbody>

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

Using HTML5 with Exhibit

Many Web authors are moving to HTML5 and Exhibit 3.0 supports the new standard. This section describes Exhibit coding changes for working with HTML5.

The Exhibit attribute-based configuration has changed for HTML5. A compatibility mode remains for Exhibits published in XHTML. HTML5 does not support XML namespaces, providing a new custom attribute mode in its stead.

Moving from Exhibit 2.2.0 in XHTML to Exhibit 3.0 in HTML5 requires changing all attributes prefixed with `ex:` to be prefixed with `data-ex-` instead. In addition, all capital letters within the attribute name should be converted to a hyphen and lower case, so for example, `ex:itemTypes` becomes `data-ex-item-types`. The HTML5 data attribute API treats capitalization differently during document processing and when attribute access occurs, necessitating the change to hyphenation.

Additional Exhibit HTML Authoring Notes

Anything used in a `rel` attribute has changed from using the / character to the - character. The / usage is deprecated; it will still work now but will not in the future. This mostly pertains to `<link rel="exhibit-data" ...>`.

HTML Attributes and Exhibit Tags

Exhibit can be invoked to provide some attributes that are found within HTML tags. This is especially useful in a lens, where data needs to be styled, formatted, or linked.

For example, suppose you are working with the `<img...>` tag, which requires the `src` attribute. To have Exhibit provide the content of the src attribute, you use `ex:src-content="..."`, substituting the ellipses with the proper expression. Here's an example:

```
<img ex:src-content=".url"></img>
```

This technique also works for the href attribute that is used with the <a...> tag. Here's an example:

```
<a ex:href-content=".url">Link text</a>
```

Additional Resources

Getting Started with Exhibit

Walk step-by-step through the basics of creating an exhibit (Scripted). Learn to add views, sort and filter with facets, add lenses and styles, and view data in a timeline.
See Getting Started with Exhibit:
http://www.simile-widgets.org/wiki/Exhibit3

Exhibit Reference Documentation

Learn more about coding exhibit HTML pages in the Exhibit Reference doc available on the documentation wiki.

Developer Documentation

See the developer documentation for Exhibit developers who want to customize or examine how Exhibit works:

Developer Doc for Exhibit 3.0 Scripted:
https://github.com/zepheira/exhibit3/wiki

Developer Doc for Exhibit 3.0 Staged:
https://github.com/zepheira/backstage/wiki

How-To Articles

The Exhibit community is very diverse and several exhibit users have written up procedures and articles for creating a variety of exhibits.

See the Simile Widgets collection of How-To Articles for articles written by authors using Exhibit and other technologies. You're invited to contribute your own articles, add tips for working with Exhibit 3.0, and share your expertise with the community.

FAQs

As Exhibit has matured, the community has amassed a store of knowledge on creating and managing exhibits.

Check the FAQ on the doc wiki to see if someone else has answered your questions already. If you work through a problem or discover a new way to use a feature, you're encouraged to add your solution to the FAQ.

Exhibit 3.0 Demos

The Exhibit 3.0 software is comprised of two separate components, **Scripted** for building in-browser Exhibits, and **Staged** for larger, server-based Exhibits.

By viewing the page source of Exhibit examples, you can learn how other authors add interactive elements to their exhibits and make the most of Exhibit's views, lenses and facets.

See the demos on the continuous integrator to see live examples of Exhibit 3.0 in Staged and Scripted modes:

Sample Exhibits: Scripted Mode

- Nobelists: live Exhibit
- Senate: live Exhibit
- Senate in HTML5: live Exhibit using new HTML5 configuration language

Sample Exhibits: Staged Mode

- The Encyclopedia of Life
- Library of Congress Prints and Photographs
- Sweden Europeana

Using Views, Facets, and Lenses in Your Exhibits

With Exhibit, you can add interactive elements to your Web page with just a few tags. This section shows how to use Exhibit code in your HTML to let users can view and interact with your data.

Views are ways of looking at collections of items. Every Exhibit uses a view to display your data. You can add and change views as you like, but your exhibit must use a view.

Facets are the properties of items that you can expose for filtering, sorting or searching on an exhibit.

Lenses are ways of formatting individual items. Views will show collections of items, often by invoking lenses to show each item in the collection.

Lenses and facets let you control which items or data properties to display, adding interactivity to the page. Both views and lenses are optional.

Views and Facets can be restricted to show objects of a certain type by using Collections. See the developer doc on GitHub for information on customizing your use of views, lenses, and facets.

For a demo of the Hierarchical view see http://databench.zepheira.com/demos/icd/.

See the feature map showing which views, facets, and lenses are currently supported in Exhibit 3.0 Scripted and Staged modes.

Views

Every exhibit uses a view to display data. The range of views you can use in authoring your exhibit depends on whether you're publishing an in-browser exhibit with Scripted mode or a server-based exhibit in Staged mode.

Exhibit 3.0 Staged:
E**xhibit 3.0 Staged mode** on Backstage currently supports the Tile View. See the developer documentation on GitHub for details on how Backstage supports views.

Exhibit 3.0 Scripted:
Views available in Exhibit 3.0 Scripted mode include Tile, Tabular, Timeline, and Thumbnail. Map views are under development.

Example Code
This sample code shows a Tabular view for the Exhibit demo showing legislation passing through the U.S. Senate.

```
 <div ex:role="exhibit-view"
 ex:viewClass="Exhibit.TabularView"
 ex:label="Table"
 ex:columns=".label, .party, .state, .committeeMember.label, !sponsor,
!cosponsor"
 ex:columnLabels="name, party, state, member of, sponsored, cosponsored"
 ex:columnFormats="list, list, list, list"
 ex:sortColumn="4"
 ex:sortAscending="false"
 ex:rowStyler="rowStyler">
</div>
```

Reference Documentation: Views
See the Reference Documentation for details on coding Views for your exhibits:

Views `<...ex:role="exhibit-view"...`

- Tile View (Default View)
- Tabular View `...ex:viewClass="Tabular"...>`
- Timeline View `...ex:viewClass="Timeline"...>`
- Thumbnail View `...ex:viewClass="Thumbnail"...>`
- Map View `...ex:viewClass="Map"...>`
- OpenLayers Map View `...ex:viewClass="OLMap"...>`
- Simple Map View `...ex:viewClass="SimpleMap"...>`

Developers looking to create customized views for Exhibit 3.0 Scripted mode should see the developer documentation on GitHub.

Views: What's New in Exhibit 3.0

Toolbox Parameter for Views

In Exhibit 3.0, the semantics of the toolbox parameter for views (ex:showToolbox) have been unified. All views now have ex:showToolbox, a boolean, set to true by default. The toolbox is displayed by default, so you don't need to hover with the mouse to reveal it.

A new parameter (ex:toolboxHoverReveal), a boolean, set to false by default, will restore the old behavior of hovering to see the toolbox.

Using the New Control Panel Bookmark Widget

Exhibit 3.0 includes a new control panel that appears by default before the first view or panel showing a bookmark UI widget. Because the bookmark feature didn't exist before Exhibit 3.0, when you move an existing exhibit to Exhibit 3.0 the control panel will appear.

The Exhibit 3.0 bookmarking tool lets user save the current Exhibit URL. The bookmark widget lets end users open a popup showing a URL they can either bookmark or share with others, for a permanent link to that view. Note that the bookmark will most likely be quite long – you might suggest that users run the URL through a shortener like bit.ly.

This section describes the code for the control panel with the new Bookmark UI widget. This tool appears by default above the first view or view panel in your exhibit.

Here's the code that controls the control panel:

```
<div ex:role="exhibit-controlPanel"
    ex:showBookmark="[true|false]"
    ex:developerMode="[true|false]"
    ex:hoverReveal="[true|false]">
```

`ex:showBookmark` is true by default. The other settings are false by default.

`ex:showBookmark` indicates whether to show the bookmark widget. The bookmark widget lets your exhibit users open a popup showing a URL they can either bookmark or share with others, for a permanent link to that view. Note that the bookmark will most likely be quite long – you might suggest that users run the URL through a shortener like bit.ly.

`ex:hoverReveal` indicates whether the control panel (and the widgets it contains, such as the scissors icon for exporting data) should be invisible unless hovered over with the mouse.

`ex:developerMode` indicates whether the control panel will display widgets for developers. This currently includes a mechanism for resetting the accumulated history a user has for the Exhibit they're viewing.

To eliminate the control panel from view, you can either give it an ID and use CSS to hide that ID, or set the `ex:showBookmark` setting to false.

Facets

An Exhibit Facet lets you restrict the viewable data set to only those items matching the facet expression and those viable values.

For example, in the sample Senate exhibit, users can choose to see only the senate bills sponsored by senators from their state, by choosing from the State selector on the right side of the page. This selector is a facet for sorting and filtering data by state.

Code Example: Facets

```
<td id="sidebar">
[…]
<div>
ex:facetLabel="State" id="represents-facet"></div>
[…]
</td>
```

Choosing Facets for Exhibit 3.0 Staged Mode

Exhibit 3.0 Staged mode, built on Backstage, implements the List Facet, though any facet value with a count of one will be elided. There are some additional considerations for choosing a facet with exhibits built on Backstage. See the Backstage documentation for details.

Facets and Exhibit 3.0 Scripted Mode

In Scripted mode, Exhibit 3.0 supports the following Facets for filtering, sorting and searching data in an exhibit.

See the Reference Documentation for details on using each type of Facet:

Facets `<...ex:role="exhibit-facet"...`

- List Facet (Default Facet)
- Tag Cloud Facet `...ex:facetClass="Cloud"...>`
- Text Search Facet `...ex:facetClass="TextSearch"...>`
- Numeric Range Facet `...ex:facetClass="NumericRange"...>`
- Alphabetical Range Facet
- Hierarchical Facet `...ex:facetClass="HierarchicalFacet"...>`

Developers looking to create customized facets for Exhibit 3.0 Scripted mode should see the developer documentation on GitHub.

Lenses

You use Lenses to customize the display to show only certain parts of a data set or item, and to customize how the data looks.

Note: Using Lenses with Exhibit 3.0 Staged Mode

Backstage implements a subset of the lens language. Exhibit 3.0 Staged on Backstage recognizes the following lens attributes:

- `if-exists`
- `*-content`
- `*-subcontent`
- `*-style-content`
- `*-style-subcontent`

Customizing Your Exhibit with a Lens

To add a custom lens template your exhibit, add

```
<div ex:role="exhibit-lens"> ... </div>
```

Settings include:

setting name	type of value	default	choices	means
ex:itemTypes	list of item-type-names	(none)		the item types that should be displayed using this lens; if unspecified, this is the default lens for all item types
ex:onshow	JavaScript or style attributes	(none)		to execute JavaScript or change style attributes when the lens is displayed

The URL of an Exhibit page followed by a hash with the label of an item will display a dialog box with that item's properties. You can create a lens template for this dialog box to specify which properties are displayed and how. The lens needs to be inside the body of the HTML page but outside any view elements, with its display style set to none. For example, this lens controls what will be displayed in a dialog for a single item of type Person:

```
<div ex:role="exhibit-lens" ex:itemTypes="Person"
style="display: none">
...
</div>
```

ex:onshow can be used to execute JavaScript or change style attributes when the lens is displayed.

Code Example: Lenses

```
 <div ex:role="exhibit-lens" ex:onshow="this.style.background =
'blue';">...</div>
```

This code causes the lens to always be blue. (Of course you could have just used style="background: blue".) Use this.getAttribute('ex:itemID') to get the ID of the item. An elaborate example would include a lens template like this

```
 <div ex:role="exhibit-lens" ex:onshow="prepareLens(this);">
 ...
 <div ex:id-subcontent="tab1-Template:Value">...</div>
 <div ex:id-subcontent="tab2-Template:Value">...</div>
 ...
 </div>
```

together with some JavaScript like this

```
   function prepareLens(elmt) {
      var itemID = elmt.getAttribute("ex:itemID");
      var tab1 = document.getElementById("tab1-" + itemID);
      var tab2 = document.getElementById("tab2-" + itemID);
      ...
   }
```

About Collections

Collections sit atop the exhibit database as a defined subset of items from within the database. By adding an exhibit-collection to the page, Exhibit can select items of a certain type. It can be put right after the body tag for example.

This lets you restrict the items displayed in an exhibit view to a specific item type or types. For example if you have both President and Event types in your JSON file, you can restrict your Tabular view (or any other view) to just the President item types using the code below.

```
<body>
  <div ex:role="exhibit-collection" ex:itemTypes="President"></div>
...
```

For Multiple Collections Being Presented

You need to give different collections different IDs:

```
   <div ex:role="exhibit-collection" id="typeA-things"
ex:itemTypes="typeA"></div>
   <div ex:role="exhibit-collection" id="typeB-things"
ex:itemTypes="typeB"></div>
```

Then link the views and facets to the right collections:

```
 <div ex:role="exhibit-view" ex:viewClass="Timeline"
ex:collectionID="typeA-things" ...></div>
 <div ex:role="exhibit-view" ex:viewClass="Map" ex:collectionID="typeB-
things" ...></div>
```

Facets can be linked the same way.

Learn more about collections in Exhibit in the developer documentation.

Collections in Exhibit 3.0 Staged

Backstage allows for subsets of the contents of its database to be grouped into collections, based on the following divisions:

- All items (default)
- Only items of one certain `rdf:type`