


Exhibit 3.0

Architecture

Document

Disclaimer

The following page is a dynamic, living document designed to be updated though out the course of the project. Comments and feedback are welcomed on the simile-widgets  discussion list.

Overview and Objectives

Exhibit enables web site authors to create dynamic exhibits of their collections without resorting to complex database and server-side technologies. The collections can be searched and browsed using faceted browsing. Assorted views are provided including tiles, timelines and maps to name but a few.

Exhibit was originally developed as part of the MIT Simile Project (simile.mit.edu), an ambitious collaboration of the MIT Libraries, the MIT CSAIL, and the World Wide Web Consortium (W3C) to explore applications of the Semantic Web to problems of information management across both large-scale digital libraries and small-scale personal collections. Exhibit is a light weight application that runs inside a web browser and allows web site authors the ability to publish and navigate data via common Web standards. As a client applications running within a Web browser however, Exhibit is limited to small-scale collections of data.

The ease in which web site authors can now publish data using Exhibit has demonstrated that authoring data-interactive sites can be as easy as authoring a static web page. With the increased interest in data-interactive sites, however, the limitation

Contents

1. Exhibit 3.0 Architecture Document
 1. Disclaimer
 2. Overview and Objectives
 3. Design Philosophy
 4. Architectural Design
 1. Exhibit 3.0 Scripted
 2. Exhibit 3.0 Staged
 3. Data Services
 4. Defining the Exhibit 3.0 Staged Client/Server Interface
 1. Backstage
 2. Refine
 5. Deployment Model/Overview
 6. Feature Definition
 7. Use Case View
 1. From Library of Congress
 2. From Simile Tools Workshop - June 2010
 8. Glossary of terms
 9. Acknowledgement

of Exhibit for addressing larger collections of data has become a barrier to wider adoption of this tool, and to the delivery, navigation and reuse of data on the Web.

The Exhibit 3.0 project will redesign and re-implement Exhibit to scale from small collections to very large data collections. The redesigned Exhibit will be as simple to use as the current tool but more scalable, more modular, and easier to integrate into a variety of information management systems and websites—offering an improved user experience. Further, this project will document, where appropriate, the supporting Exhibit libraries and APIs to allow 3rd party application developers and others to more effectively contribute and utilize Exhibit services. The goal of this project will be to provide libraries, cultural institutions and other organizations grappling with large amounts of digital content, with an enhanced tool that is scalable and useful for data management, visualization and navigation.

Design Philosophy

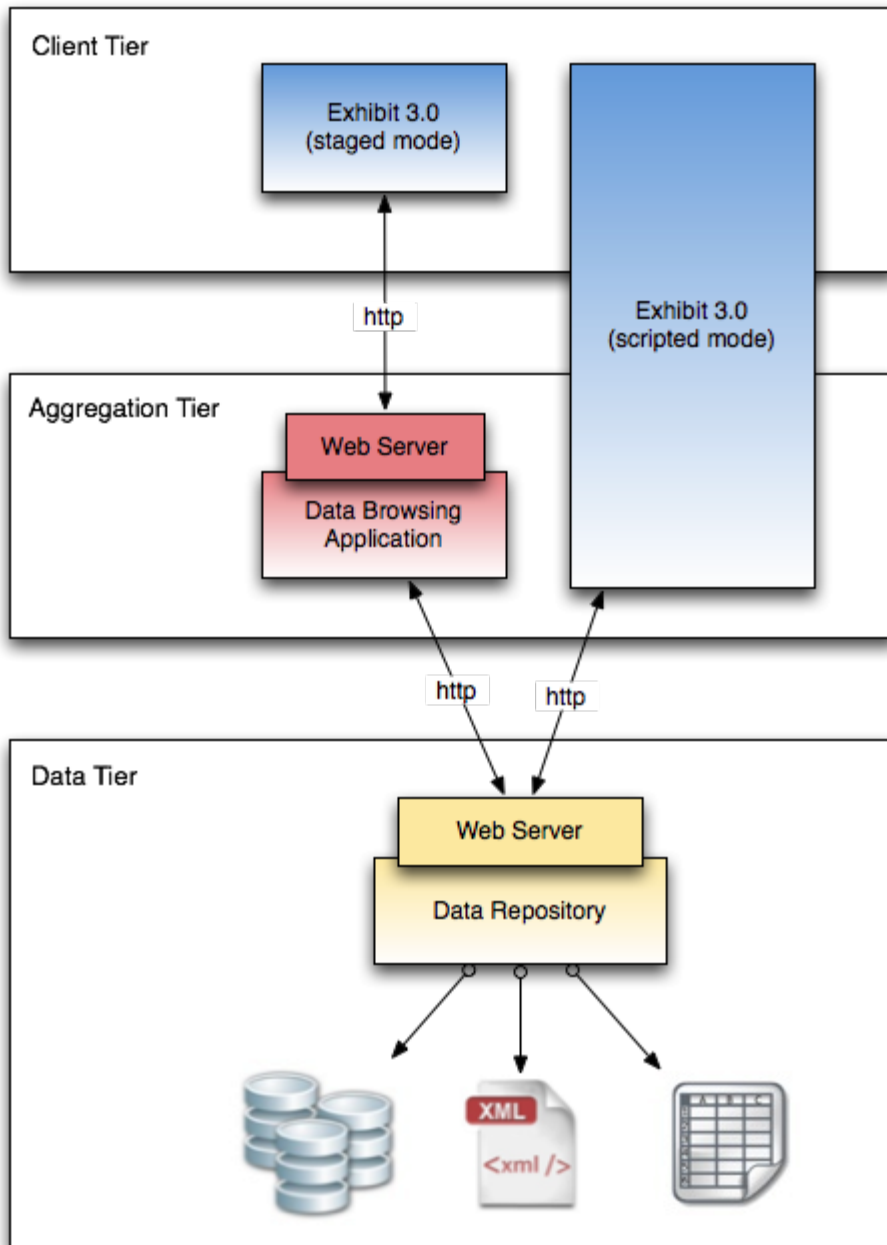
The central objective is to produce a tool capable of widespread deployment for production use - not as a proof-of-concept/prototype/demonstrator. Thus, although Exhibit3 builds upon many of the SIMILE project's outputs, it does not represent a continuation of a research agenda. Rather, it is an attempt to engineer solutions for a discrete number of limitations in the scalability and extensibility of the current codebase, using existing, well-understood technical approaches.

The deployability goal also means that Exhibit3 will not necessitate special or advanced client platform infrastructure beyond what standards-compliant web browsers offer today. For example, non-standard, or HTML5-specified - but not yet widely supported - features should not be required for a solution. The production-readiness also means that there will be a bias in favor of performance, stability, and maintainability over correctness or completeness of functional scope, where there are conflicts. While full cross-browser unit testing is outside the scope of this project, targeted browsers will support jQuery 1.5+ and include Firefox 3+, Chrome 6+, Opera 10+ and Safari 4+ and IE 8+. Additional browsers released during the course of this work (e.g Firefox 4+, IE9, etc.) will be tested as appropriate. Our goal is to leverage cross-browser tested javascript and css libraries where applicable to help achieve a consistent user experience across a wide range of browsers.

Architectural Design

Exhibit will operate in two architecturally separate but related modes, one run entirely

by the user-agent (**scripted**) the other a client-server model (**staged**) where the user-agent primarily deals with user interaction and coordinates what is displayed with the server. The Exhibit 3.0 **scripted** client does not require any server backing its operation and is intended for use with limited size data sets. The following diagram reflects this architecture.



In both cases, the data Exhibit displays can come from several services, from static files to cloud-based data warehouses.

Exhibit 3.0 Scripted

In **scripted** mode, all of the data and the software scripts are transmitted to the user-agent where the scripts are interpreted and run by the in-browser JavaScript engine. The displayed data is cached, indexed, and queried in the user-agent, and all computation after initial transmission is accomplished by the user-agent. This mode will reflect the current model found in Exhibit 2.0.

This approach is limited by the computational power of the user's computing environment. Only a limited size data set can be viewed in Exhibit before constraints on processing power, memory, and user tolerance of performance win out. At that point, intervention by the creator would be needed to explicitly switch to Exhibit **staged** mode.

Exhibit 3.0 Staged

In **staged** mode, Exhibit behaves as a client / server. The software scripts are transmitted to the user-agent where they are employed to listen to user actions and translate them from context into queries to send to the server. The displayed data is stored, indexed, and queried on the server. Stateless queries received by the server are computed and responses returned to the querying client which then modifies the user-agent display accordingly.

This approach is less constrained by the user's computing environment and may still be performed on older hardware. The server becomes the limiting factor, and as a stateless, public service must support multiple clients issuing multiple queries simultaneously over data sets too large for the Exhibit 3.0 **scripted** in-browser client to render efficiently.

Data Services

While some Exhibit designers may choose to serve their own curated data sets, in either mode, both modes may also utilize external data services as a way of staying in sync with a canonical data source or for bringing disparate data together into one Exhibit view. Exhibit 3.0 in staged mode, or the Exhibit server (in support of scripted mode) will read from one or more services using applicable query languages to communicate.

Defining the Exhibit 3.0 Staged Client/Server Interface

One of the end products of the project will be an agreed contract between client and server. We examine existing implementations below for ideas on how to proceed.

Backstage

SIMILE Backstage, written by David Huynh to sit atop the SIMILE Butterfly service, written by Stefano Mazzocchi, came the closest to fulfilling some of the Exhibit 3.0 Project goals during the SIMILE Project. Its architecture is examined below for some ideas on how Exhibit might be structured.

However, at a high level, it should be noted one of the implicit goals of Backstage was to incorporate the previous Exhibit software into its structure without rewriting it. While this was done quite successfully, due in part to David's earlier design decisions, we have the opportunity to start without such constraints and should take advantage of it accordingly. For instance, the calls made between the client and server in Backstage could be abstracted to be the same set of calls in both the staged and scripted client modes of Exhibit, just implemented differently. Backstage currently brings in the entire Exhibit code base, then redefines methods to be compatible with Backstage's API; eliminating previously unavoidable inefficiencies like this would be helpful.

A key interface specification for client-server communication can be found in the [Backstage repository](#). All interactions involve an HTTP GET of one server URL with variable query arguments, one of which is a payload containing all the parameters specific to the method being invoked.

There are only a handful of defined methods:

Method	Description
<code>initialize-session</code>	Creates a server-side representation of an Exhibit
<code>add-data-links</code>	To the server-side representation, add all URLs to data shown in the Exhibit
<code>configure-from-dom</code>	To the server-side representation, add all the configuration associated with the Exhibit
<code>facet-apply-restrictions</code>	Calculates and returns the facet listing when a new constraint is selected in the Exhibit
<code>facet-clear-restrictions</code>	Calculates and returns the facet listing when constraints are removed in the Exhibit

<code>generate-lens</code>	Calculates and returns the lens contents after a change in selection is made
----------------------------	--


Backstage only defines the list facet and the tile view as proof of concept.

Refine

Differences

- Refine's primary strength is in editing, which Backstage and Exhibit don't have.
 - As collaborative editing is too difficult, we assume single-user usage for the early versions (whereas Backstage is supposed to be multi-user). This means the Refine code isn't so rigorous on locking and such.
 - Editing requires certain operations, like search and replace by regex, that is not of use in a read-only browser like Exhibit or Backstage.
- Refine supports grid-shaped data natively, not graph-shaped data like Backstage and Exhibit, for performance reason. The implication is that its expression language does not conveniently support graph traversal as Exhibit's language does.
- Refine currently does not support views and lenses while Backstage does. Lenses can be tricky to implement.
- In Refine, we go easy on the support of the Back button: clicking the Back button does not undo anything, whereas in Backstage and Exhibit, it should undo the last view or facet action.
- Refine is assumed to run local with very low latency, so its UI components are allowed to update independently with little fear of them looking out of sync at any moment. Backstage was intended to run on a server, with high latency, so updates to its UI components are highly orchestrated so that they don't appear out of sync. This added complexity to Backstage's UI code.
- Refine's backend is supposed to support its own front end only, at the same URL domain, whereas a Backstage server might serve (through JSONP) embedded clients in other URL domains. This means Refine can do POSTs and Backstage is stuck with JSONP GETs (meaning, smaller HTTP payloads traveling on URL parameters).
- Refine is not embedded in some larger site and does not need to worry about CSS style collision, or conflicts with other ajax frameworks.

David Huynh's review and input especially welcome here; any thoughts comparing Refine to Backstage also welcome

 Google Refine provides "a power tool for working with messy data," with a purposes slightly different from that of this project's goals. The interface is based on a tabular representation with facets and other editing and reconciliation tools made available.

Deployment Model/Overview

The ability to switch between Exhibit 3.0 scripted and staged mode is specified in HTML and similar to the mechanisms used in Exhibit. An objective of this project is to be able to leverage the same Exhibit layout and display mechanisms for either mode.

```
<!-- Example of scripted client invocation -->  
  
<script src="http://api.simile-widgets.org/exhibit/3.0/exhibit-scripted-api.js" type="text/javascript"></script>  
<link href="data.js" type="application/json" rel="exhibit/data" />
```

```
<!-- Example of staged client invocation -->  
  
<script src="http://api.simile-widgets.org/exhibit/3.0/exhibit-staged-api.js" type="text/javascript"></script>  
<link href="data.js" type="application/json" rel="exhibit/data" />
```

The example URIs used for the Exhibit 3 javascript end points are included here for illustrative purposes. The active Exhibit 3 identifiers along with specifics of the adaptive parameter variables and permitted values for Exhibit 3.0 staged mode will be refined and documented over the development phase of this project.

Feature Definition

Note: "Deferred to Later Phase" section will be refined throughout the course of the project.

1. Scalability


Exhibit was designed as a Web browser widget that stores all of its data in memory, so it can only handle a small number of items (around 500 to 1,000 depending on the complexity of the items). Since many Exhibit users have much larger collections that they want to put in Exhibit, it will be necessary to redesign the tool to support both small and large collections. The scale of collection that we require for Exhibit 3.0 is

characterized below:

- Number of items $\leq 100K$
- Number of triples $\leq 1m$
- Number of bytes/triple $\leq 1K$
- Number of facets ≤ 20
- Number of facet values $\leq 2K$

DEFERRED TO LATER PHASE:

- Scaling to a larger number of items in a specific view (e.g. map, timeline)
- Number of item types or horizontal scaling (e.g. to enable link sliding)

 Development Discussions and Feature Status: Scalability

2. Extensibility (i.e. new views/facets/widgets)

Exhibit was designed to be a data navigation and visualization framework into which new views of data (e.g. plot, graph, map, timeline, or a particular facet) could be added. This process is working but with extreme difficulty, partly because it's not well explained how to do it, and partly because the code to support this is complex. We agreed to the following priorities for Exhibit 3.0:

- Document the API for writing new view/facets/widgets with clear examples
- Make it easier to connect new views/facets/widgets to the framework
- Add an API to support result set pagination and/or summarization
- Add support for adding/subtracting items from a view/facet/widget incrementally

THIS PHASE:

- Simple list facet (highest priority)
- Default list view (highest priority)
- Other facets: Tag cloud, search, sliders (Further evaluation required to determine list in scope for this phase. To be completed as we can manage given timeframes and depending on current state of code.)
- Other views: Map, timeline, and table views (Further evaluation required to determine list in scope for this phase. To be completed as we can manage given timeframes and depending on current state of code.)

DEFERRED TO LATER PHASE:

- Facets and views from the list above which are not completed in this phase of work

Development Discussions and Feature Status: Extensibility

3. *Persistence*

As a Web browser widget, Exhibit allows users to work with the data in memory and change the local state, for example selecting facets to limit the record set they're viewing, or choosing a particular view of the data. Users want to be able to point to the finished product – the state of the Exhibit that they reached via a set of operations on the data. We agreed that Exhibit 3.0 should:

- Improve permalink (bookmarking) capability for end users to specific views with state; primarily a documentation issue

DEFERRED TO LATER PHASE:

- N/A

Development Discussions and Feature Status: Persistence

4. *Modularity (aka “embed-ability”)*

Many Exhibit users need to embed Exhibit in a software tool chain and/or a separate system (e.g. VIVO, Sakai, WordPress). This is possible but difficult to do now, so Exhibit 3.0 should provide easier ways to call Exhibit modules and address individual components (e.g. views) in separate web pages.

DEFERRED TO LATER PHASE:

- Multiple Exhibits embedded inside of the same page

Development Discussions and Feature Status: Modularity

5. *Data indexability/exportability*

Exhibit was designed to be dynamic, importing the data and JavaScript code to render it from a source outside the Web page. This makes it difficult for external service providers like Google to harvest the data for indexing etc. without extra effort by the Exhibit author. Furthermore, because its current state is only known to the local Web browser, it's difficult to point another user at an Exhibit in a particular state (i.e. offer Permalinks that take you to exactly what the user is experiencing in their current

session). Both of these should be supported by Exhibit 3.0. Specifically,


- Exhibit 3.0 should have the capability of exporting the original dataset (in pre-JSON encoding, where relevant) via a URL, such that tool chains can extract data for external processing before or after rendering, and
- Offer a permalink for the current state of the Exhibit
- Offer a documented way of encoding the data in the HTML (e.g. RDFa) that supports indexing. In addition, the documentation should explain the existing ways to support external indexing by search engines (e.g. including the data in a non-displaying HTML tag in the rendered Exhibit).

THIS PHASE:

- Export formats: JSON and CSV
- Investigate appropriate solution with search engine providers and W3C's Web Accessibility Initiative.

DEFERRED TO LATER PHASE:

- All other data formats available for export

 Development Discussions and Feature Status: Data indexability/exportability

6. Usability

Usability covers a range of behavior of the software. Here we identify some general goals for Exhibit 3.0 usability in three dimensions:

- Latency will be reasonable, e.g. under 10 seconds to render and include a progress bar
- Exhibit authors will still be able to cut and paste existing HTML to create new Exhibits, and do not need to know programming languages to create simple customizations.
- There will be an easy process to scale from small Exhibits to large ones for the same dataset as it grows over time. The transition from Web-browser only (i.e. the current Exhibit) to the server-side version of Exhibit should be simple and seamless.

DEFERRED TO LATER PHASE:

- Formal usability study

7. *Editing*

An ongoing need of Exhibit users, including both data owners who publish their data with Exhibit and end users who interact with data in Exhibit, is to edit the data directly in the tool rather than in a separate program that can open the Exhibit data file stored on the server. We recognize two distinct needs:

- Editing “master” data (usually by the data owner) in a way that keeps the changes permanently.
- Editing a copy of the data for local manipulation (e.g. a student working with a dataset as part of a class assignment) and which does not require persistence to be handled by Exhibit itself.

We recommend that simple edits (e.g. fixing a typo in a single facet value) should be handled by an Exhibit widget for editing-in-place with persistence. The MIT CSAIL Research Assistant will develop prototypes for this, for example the Dido work, described in detail on the CSAIL website¹.

More extensive editing (e.g. fixing all instances of a facet value across a large dataset) should be done outside of Exhibit, for example using the Gridworks 2 open source software tool. We did not resolve the question of where how persistence should be handled for the second case (the data copy) nor how the two cases might interoperate. Support for end user layout editing, e.g. views or facets, with persistence, is desirable but out-of-scope for now. For large dataset editing, Exhibit could support single RESTful writes back to the server but that is not a priority for Exhibit 3.0.

THIS PHASE:

- TBD as defined by MIT CSAIL

DEFERRED TO LATER PHASE:

- TBD as defined by MIT CSAIL

8. *Data Pipelining*

Exhibit uses a specific set of conventions on top of JavaScript Object Notation (JSON) as data input, and provides limited data output tools (the "scissors") actuated from the

user interface. In order to integrate more readily with the variety of potential Exhibit 3.0 environments, including LC's target Recollection platform, this project will include enhancement of data input as well as data output. Currently Zepheira has worked with LC to integrate a data pipeline based on the Akara open source (Apache 2.0 license) tool for Recollection. Akara as well as the Simile Babel tool will guide the architectural guideline for enhanced data pipelining.

- There will be an HTTP/REST-based plug-in system for data input to Exhibit 3.0, similar to the current Babel tool, but with enhanced functionality and API for extension.
- There will be an HTTP/REST-based plug-in system for data output from Exhibit 3.0, including menu updates for user actuation, expanding upon the current "scissors" concept.

THIS PHASE:

- Export formats: JSON and CSV
- RESTful endpoint to the data. Not only ability to cut and paste data, but provide a link that would allow user to access the data. Also provide a link to the entire data set (not just portion of data set shown) if appropriate.

DEFERRED TO LATER PHASE:

- All other data formats available for export
- Performance optimizations for accessing the entire data set

 Development Discussions and Feature Status: Data Pipelining

Use Case View

Use cases from which the current scope of work was derived pointed to the following themes:

Scalability

- Display: Properly render large data sets (10s of thousands to 100s of thousands of records) in list view as well as other views such as maps, tables, timelines.

Flexibility for Widgets

- Easy to incorporate new widgets

- Desire to set preferences on bubble sizes, background colors, titles, zoom ratio within the widgets for a specific data view

Maintain State:


- Maintain the state of a page as user selections are made. Remember these so that hitting the back button results in a return to this state rather than a return to defaults.

Data Portability:

- Ability to export data from an Exhibit view in multiple formats (JSON, RDF/XML, delimited text) and/or original format.

The use cases and issue tickets linked below provide more details of requirements as specified in the context of specific projects.

From Library of Congress

-  Exhibit limitations discovered during development of Recollection Phase 4


From Simile Tools Workshop - June 2010

-  MIT FACADE
-  Sakai
-  Simile Tools
-  Zepheira

Each use case needs to be evaluated by its owner and determine whether it can be shared publicly. Slide decks should be translated to a standard use case format before sharing.

Glossary of terms

Acknowledgement

This document has benefited from inputs from many members of the  Exhibit 3.0 project. Specific thanks are due to Adam Marcus, David Feeney, David Huynh, David Karger, Edward Benson, Eric Miller, Kathy MacDougall, Mackenzie Smith, Mark Baker, Richard Rodgers, Ryan Lee, Stefano Mazzocchi and Uche Ogbuji who have

provided (or will provide ;)) valuable contributions to this document.

ZExhibit3: Architecture (last edited 2011-04-06 08:39:20 by EricMiller)