Bachelor thesis

Creating interactive web pages using the Exhibit framework

carried out by

Josef Dabernig
Engerthstrasse 110-118/11/22

A-1200 Vienna, Austria

Assessor: FH-Prof. Dipl.-Ing. Harald Wahl

Vienna, June 9th, 2008

carried out at the
University of Applied Science Technikum Wien
Study course Computer Science

FACHHOCHSCHULE
TECHNIKUM WIEN

Abstract & Kurzfassung

Abstract

This thesis focuses on the open source JavaScript framework, Exhibit. Since 2006, it allows users to create data-centric interactive websites, so called *exhibits*. Minimal setup efforts quickly lead to a visual appealing results powered by AJAX features like live filtering, sorting and grouping. When creating an *exhibit*, the user also can choose between various data formats like spreadsheets, TSV or RDF/XML.

In “Creating interactive web pages using the Exhibit framework”, the reader will learn what Exhibit is, how it works and how to create a first *exhibit*. The thesis also aims to list and extend technical documentation, available on Exhibit. This is realized by providing a complete listing of existing documentation resources and by using Exhibit itself to demonstrate, how documentation available on Exhibit can be extended. Afterwards, the demonstration of two real-world use cases shows, that Exhibit also fits into database-driven, three-tier web applications.

Users of Exhibit without any programming skills may learn about the basic concepts in a learning-by-doing process accompanied by source code examples and their explanation. Developers familiar with web design technologies find a technical documentation reference and additional examples of using Exhibit in diverse environments.

Kurzfassung

Die folgende Arbeit beschreibt das Open Source JavaScript Framework Exhibit. Seit 2006 ermöglicht es Benutzern die Erstellung sogenannter *exhibits* - interaktive Websiten zur Präsentation von Daten. Erste, visuell ansprechende Ergebnisse sind bereits nach einem minimalen Konfigurationsaufwand möglich und beinhalten AJAX Funktionalitäten wie Filterung, Sortierung und Gruppierung in Echtzeit. Beim Erstellen von *exhibits* kann der Benutzer ausserdem zwischen verschiedenen Datenformaten wie einer Excel Tabelle, TSV oder RDF/XML wählen.

„Creating interactive web pages using the Exhibit framework” vermittelt Sinn und Zweck von Exhibit, seine Funktionsweise und wie ein erstes *exhibit* erstellt werden kann. Ziel der Arbeit ist auch die Auflistung und Erweiterung technischer Dokumentation bezüglich Exhibit. Hierzu dient einerseits eine Referenz bestehender Dokumentationsressourcen, als auch der Anwendung von Exhibit selbst zur Darlegung weiterer Möglichkeiten, die vorhandene Dokumentation zu erweitern. Weiters folgen Demonstrationen zweier realer Beispiele, welche aufzeigen, dass Exhibit auch im Umfeld einer Drei-Schichten-Architektur mit Datenbankanbindung eingesetzt werden kann.

Benutzer von Exhibit ohne Programmierkenntnisse erlernen grundlegende Konzepte im Zuge praktischer Anleitungen, kombiniert mit Quellcode und dessen Dokumentation. Erfahrene Web-Entwickler finden eine Referenz zu technischer Dokumentation und weitere Beispiele der Anwendung von Exhibit in unterschiedlichen Umgebungen.

Statutory Declaration

“I declare that I have developed and written the enclosed thesis entirely by myself and have not used sources or means without declaration in the text. Any thoughts or quotations which were inferred from these sources are cleary marked as such. This thesis was not submitted in the same or in a substantially smiliar version, not even partially, to any other authority to achieve an academic grading and was not published elsewhere. “

Vienna, June 9th 2008
Josef Dabernig

Table of Contents

1 Problem & Objectives

Exhibit is an open source tool, which aims to simplify data publishing processes on the web.

This thesis was written because of a personal believe, that Exhibit both brings good concepts and a proven implementation. Being an open source framework which enables publishers with low technical skills to provide their data in a feature-rich way, it fulfills a desire of equalization to the world wide web as an open medium.

1.1 Requirements

At the time of writing, Exhibit is a quite stable framework, but compared to large open source projects like Firefox or Ubuntu, it is much smaller. In comparison to such large projects, Exhibit is behind in the size of its community, the number of contributors and the availability of documentation. Due to this facts, this thesis was written, in order to fulfill the following requirements:

- Explain purpose and basic concepts of Exhibit
- Illustrate Exhibit's usage by examples
- Extend documentation, available for Exhibit

1.2 Target Groups

“Creating interactive web pages using the Exhibit framework” focuses on two target groups:

- **Users** without any or with limited programming skills may learn from the advantages of using Exhibit to publish their data in chapters *3* and *4*.
- **Developers**, familiar with web development technologies like HTML, JavaScript and JSON, additionally can find technical documentation on Exhibit in chapters *5* and *6.*

Note, that chapter *6 Real-World Use Cases based on Exhibit* also investigates the programming languages Java and Python and the web development frameworks Zope and Plone. In order to completely understand the examples described, a basic knowledge of these technologies would be helpful.

1.3 Approach

The approach taken, was reading documentation, currently available on Exhibit. Missing parts where interpreted from or even requested at the newsgroup, where Exhibit's core developer, David Huynh, eagerly supports users and contributors of Exhibit. Additionally, the source code itself and SVN commits where analyzed and Javascript debugging using the Mozilla Firefox extension Firebug also helped a lot understanding, what's going on.

1.4 Chapter Overview

To fulfill the stated requirements, the thesis consists of several **parts**:

- *Chapter 2 Introduction to Exhibit* covers a short overview of the basic intentions of the Semantic Web, the existence of a Semantic Web-dedicated project named SIMILE and homed at the MIT and its sub-project, Exhibit.
- *Chapter 3 Basics of Exhibit* provides an overview of the intentions and basic concepts of Exhibit, a short summary of its historical background and describes the architecture, Exhibit is built on.
- *Chapter 4 Usage of Exhibit* takes Exhibit into action by examining the creation of a basic *exhibit*. Further, possibilities of adding advanced features to the created *exhibit*, by using Exhibit's core functionalities or available API Extensions are discussed.
- *Chapter 5 Technical Documentation of Exhibit* includes a listing of sources of information, available on Exhibit. It further discusses the topic API documentation and presents a prototype “Exhibit Code Documentation”, realized using Exhibit.
- *Chapter 6 Real-World Use Cases based on Exhibit* shows practical use cases based on Exhibit.
- *Chapter 7 Conclusion & Discussion* summarizes the thesis and reflects on the question, if the thesis is able fulfill its requirements. Also, the future of Exhibit and its relation to other ongoing projects are discussed.

2 Introduction to Exhibit

The introduction covers a short overview of the basic intentions of the Semantic Web, the existence of a Semantic Web-dedicated project named SIMILE and homed at the MIT and its sub-project, Exhibit.

2.1 Semantic Web

The World Wide Web Consortium (W3C) published numerous specifications, publications and presentations dedicated related to the common topic “Semantic Web Activity”. On its dedicated web site, the W3C introduces the Semantic Web by the following sentence:

*“The **Semantic Web** provides a common framework that allows **data** to be shared and reused across application, enterprise, and community boundaries”*

(World Wide Web Consortium, 2008a)

A sequent introduction of the topic reveals, that much data is kept within applications and can't be accessed from outside today. This is because a lack of semantic readability of the data. The Semantic Web is a collection of approaches that enforce semantics on data. Thus the data would become reusable and may be interpreted, reused and transformed by third party applications. (World Wide Web Consortium, 2008b)

2.2 SIMILE Project

The SIMILE Project is an umbrella project conducted by the Massachusetts Institute of Technology (MIT), which stands for "*Semantic Interoperability of Metadata and Information in unLike Environments*". SIMILE consists of numerous open source projects, representing tools in the context of semantic-web-enabling technologies. (SIMILE Project, 2008 and STEFANO, 2007)

This thesis is about Exhibit, one of those sub-projects homed at SIMILE. Exhibit itself builds upon other SIMILE projects. For example it uses Babel as a service for real-time conversion between various data formats.

2.3 Exhibit

Exhibit is "*a very lightweight AJAX Framework that lets individuals who know only basic HTML create web pages containing rich, dynamic visualizations of structured data and supporting faceted browsing and sorting on that structured data*" (HUYNH *et al*. 2007, p.1). This means, that Exhibit was developed for the "average user". By requiring minimal setup, it supports the creation of data-centric websites, which are able to compete with professional ones in terms of visual appeal. (HUYNH 2007a, p.17).

As chapter 6 Real-World Use Cases based on Exhibit demonstrates, Exhibit's features also fit well for rich presentation of data in complex web applications. Although its main focus on smallish sets of data implies a limit in terms of scalability. Chapter 7 Conclusion & Discussion deals with topics like this one and also provides a look into possible, planned and ongoing improvements on the Exhibit framework.

The following screen shot shows the Exhibit web site, including a mission statement and browseable example *exhibits:*

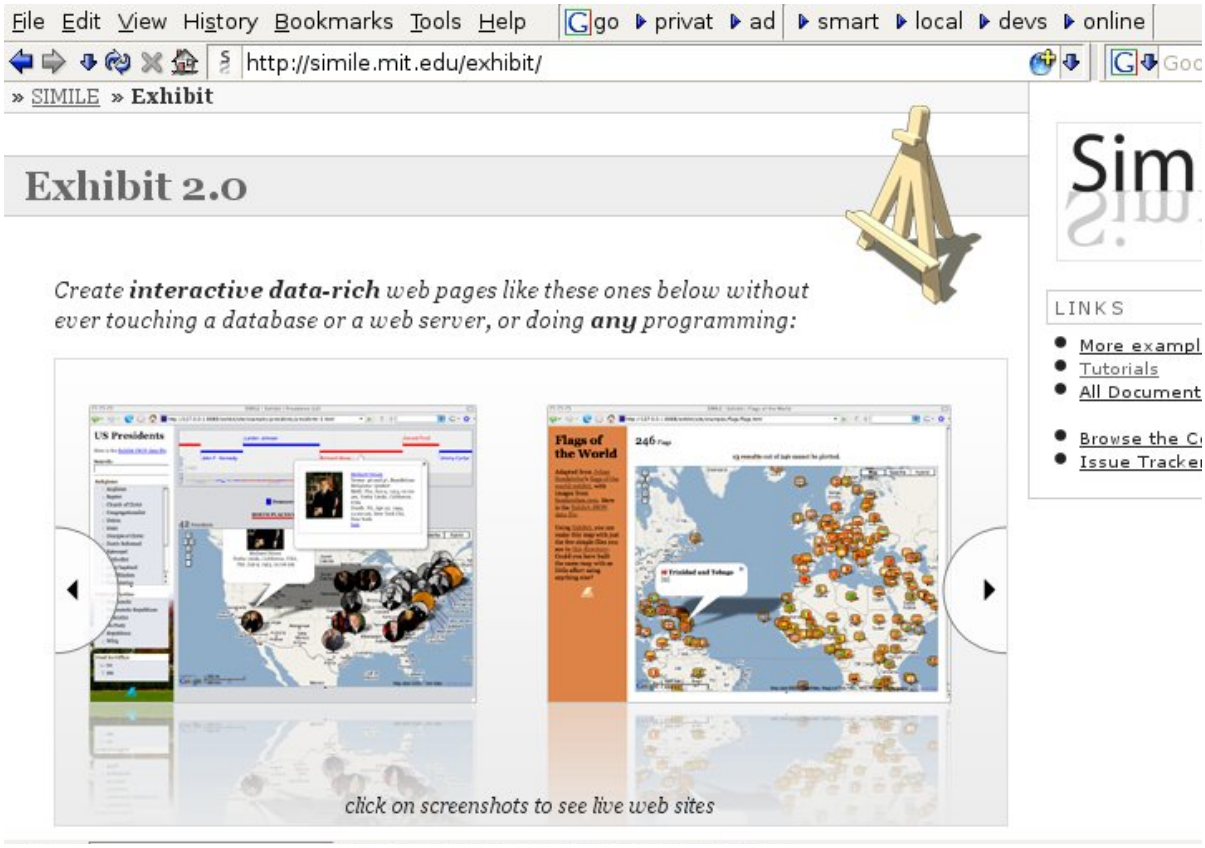


Illustration 1: Exhibit web site hosted by SIMILE, showing two example exhibits

3 Basics of Exhibit

This chapter provides an overview of the intentions and basic concepts of Exhibit, a short summary of its historical background and describes the architecture, Exhibit is built on.

3.1 Intentions and Basic Concepts of Exhibit

David Huynh (2007, p.35) states in his thesis "User Interfaces Supporting Casual Data-Centric Interactions on the Web", that Exhibit "*targets casual users instead of large publishers*". His principal intentions are described the beginning of part 3, "Publishing Data". The stated chapter describes Exhibit in a very detailed way. A summary of the intentions and basic concepts behind Exhibit, stated by Huynh in his thesis might look like the following:

Professional web site features like *searching, sorting, filtering, comparison, maps*, etc. have become a standard in todays websites. Large publishers have the resources to build feature-rich websites, but small publishers don't. Exhibit aims to provide small publishers a tool set which allows them to visualize their data in a visual-appealing way, including professional site features, as stated above, out-of-the-box. The publishing process based on Exhibit should support separating the data from its presentation and allow the publisher to edit the data in the personally preferred format. As the introductory section of "Publishing Data" ends, "*these ideas have been built into the Exhibit lightweight data publishing framework, packaged as a Web API*".

(HUYNH, 2007a, p.47ff)

3.2 History of Exhibit

Checking the history of SIMILE's publicly accessible SVN repository, David Huynh committed the first development version of Exhibit on August the 27th 2006. At this point of time, the project was named "rubik". But some weeks later, its name was changed to Exhibit, due to trademark and copyright issues, as stated in the SVN logs. (revisions 4885 and 5009 of http://simile.mit.edu/repository/exhibit)

The first version of the Exhibit site at the SIMILE wiki has been published on 20th of September 2006 and already stated the intentions for Exhibit:

"The goal of this project is to enable 'data artists' to show off their cool and tale-telling data to world-wide viewers by embedding 'exhibits' in their web pages and configuring these exhibits to best illuminate the data." (HUYNH, 2006)

Until May 2008, the SIMILE wiki received a total number of 126 pages containing documentation on the Exhibit framework (Massachusetts Institute of Technology, 2008a). On page 75 of David Huynh's thesis, "User Interfaces Supporting Casual Data-Centric Interactions on the Web", the main author of Exhibit states that 8 months after the release of Exhibit 1.0 there where more than 800 web pages using the Exhibit framework. (HUYNH 2007a, p.75)

Development of Exhibit 2.0 was started in the first months of 2007 and a page named "Exhibit/2.0 Release" exists on the SIMILE wiki, which was released on 26th August 2007. (revision 6226 of http://simile.mit.edu/repository/exhibit/branches/2.0 and Massachusetts Institute of Technology, 2008b)

3.3 Architecture

David Huynh's thesis provides information on the architecture of Exhibit in two parts:

1. The bigger part focuses on the **interface level**. It provides information on which functionalities are available and how to use them. (HUYNH, 2007a, p. 49ff)
2. A shorter chapter describes on **implementation level**, the technical realization of the features, described on interface level (HUYNH, 2007a, p. 68)

The following illustration, adapted from the thesis, gives an overview of Exhibit's architecture:

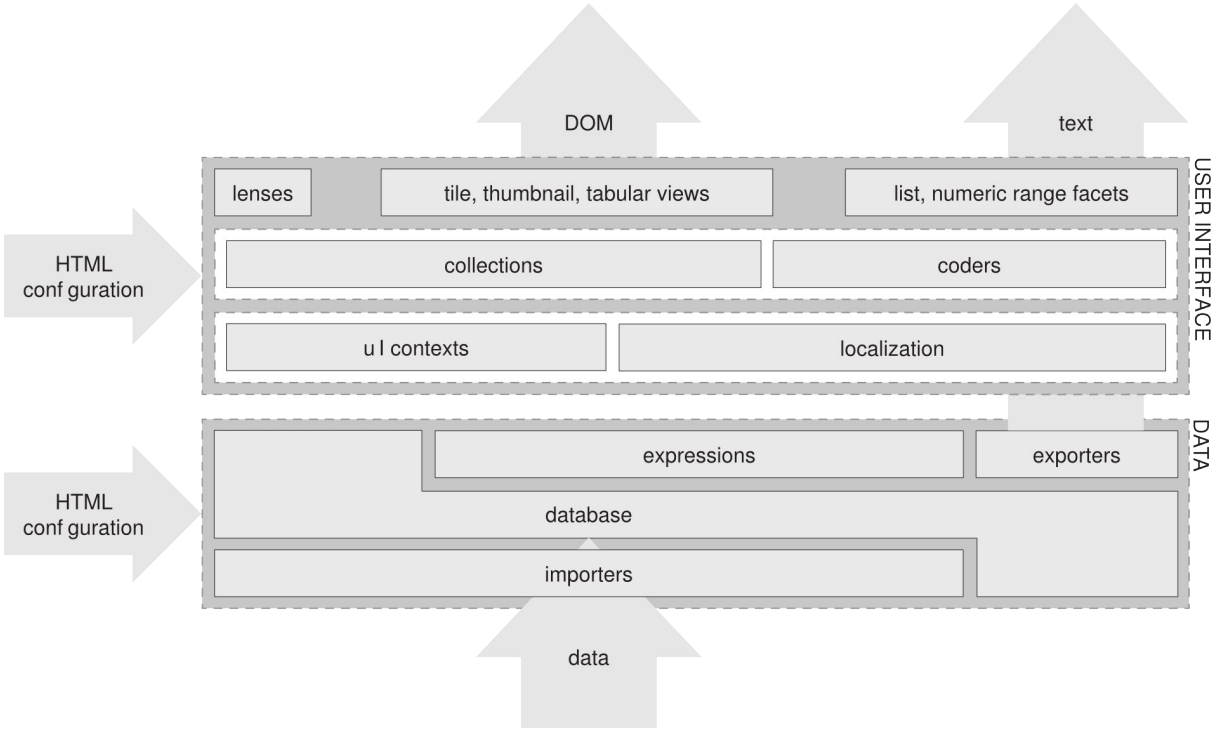


Illustration 2: Exhibit's architecture (HUYNH, 2007a, p. 69)

At the bottom is the data layer consisting of the database, the expression language parser and evaluator, and importers and exporters. At the top is the user interface layer, which consists of three sub-layers:

- *UI contexts and localization resources—storage of presentation settings for the rest of the user interface layer.*
- *collections and coders – components that do not render to the screen but determine what data widgets should render and how to render it.*
- *widgets which perform the actual rendering and support interactions.*

(HUYNH, 2007a, p. 68)

4 Usage of Exhibit

In here, Exhibit will be taken into action by examining the creation of a basic *exhibit*. Further, possibilities of adding advanced features to the created *exhibit*, by using Exhibit's core functionalities or available API Extensions will discussed.

Again, David Huynh's thesis provides in-depth information on the creation of an *exhibit*. Steps which are already explained in his thesis in detailed way, will be mentioned in a minimal way to avoid inventing the wheel twice while still giving sufficient information to get first-time users started.

The first use case is the creation of an interactive, personal curriculum vitae. On the Exhibit website (Massachusetts Institute of Technology, 2008c) one may read, that creating a feature-rich, interactive website for structured data using Exhibit is easy and wouldn't require any programming skills. Exhibit's *publisher interface* will be used to accomplish this task. David Huynh describes it in his thesis (HUYNH, 2007a, p.53ff). According to the introduction of the *publisher interface*, creating an *exhibit* requires two steps: creating the data and creating its presentation.

4.1 Starting with a minimal Setup

The first part of implementing the personal CV is the creation of a minimal setup. The minimal setup includes creation and preparation of the data for the *exhibit*. Next, a presentation will be configured and the minimal setup is finished by examining the first results.

4.1.1 Creating the Data

The data for the personal CV is already existent and stored within a local spreadsheet. Exhibit is able to read data from a Google Spreadsheet, a process which is described at the SIMILE wiki.

The following table shows an abbreviation of the data, the Google Spreadsheet contains:

{id}	{type}	{label}	{from:date}	{to:date}	{place}	{position}	{field}	{location}	{hours}	{link:url}
1	volunteer	Auslandsdienst	2008-08-01	2009-07-31	Mundos	Volunteer	Service	Nicaragua	40	http://auslandsdienst.at
2	work	Systems	2008-03-01	2008-06-13		Employee	OpenEvents	Vienna	38.5	http://smart-infosys.com
15	project	OpenEvents.at	2008-02-20	2008-04-30		Worker	Development	Vienna		http://openevents.at
16	project	Radlwolf.at	2005-09-20	2006-01-30	Dabernig	Leader	Management	Vienna		http://www.radlwolf.at
20	website-launch	The Mystery	2008				Development			http://www.the-mystery.at

Illustration 3: Google Spreadsheet data for the personal CV, abbreviated

The first row specifies the property names for Exhibit, which have to be wrapped in "{ }". The `from`, `to` and the `link` properties also specify their value types. For example "`{from:date}`" specifies, that the `from` property will be a `date`. (DUOSHUTE, 2008).
All sequent rows represent the CV data items. Every item has to define a unique `id`, if the `label` can't be ensured to be unique. Different `types` distinguish between `work` employments, `projects`, `website-launches` and so on. The `from` and `to` properties define start and beginning of the work or project. *`Website-launches`* will only contain a start date. Other properties like `place`, `position` and `field` add further description to the item and a `link` of `type` url refers to a related web site.

Generating the URL, Exhibit needs to import data from the Google Spreadsheet, is a bit tricky: The "Publish" button opens a side bar which contains a link to "More publishing options". Selecting the File format "RSS" and pressing the "Generate URL" button produces

an URL. Changing the `alt=rss` parameter to `alt=json-in-script` results in the correct URL which provides the personal CV data in JSON format.

> A published read-only version of the personal CV is available at http://spreadsheets.google.com/pub?key=pYUqUwUxZGmlikD4CPEt-WA
>
> The URL to the JSON export of the personal CV is http://spreadsheets.google.com/feeds/list/o07573463909577017953.6376616843510976610/od6/public/basic?alt=json-in-script

4.1.2 Creating the Presentation

The following lines of code contain the minimal required configuration to get an *exhibit*.

Note, that it even contains optional parts, as the page title information, an inline-style sheet and an div-element wrapping the Exhibit *ViewPanel*.

```
  <html>
    <head>
      <title>Josef Dabernig - CV</title>

      <!-- link to cv data, provided by a google spreadsheet -->
      <link rel="exhibit/data" type="application/jsonp"
  href="http://spreadsheets.google.com/feeds/list/o07573463909577017953.637661684
3510976610/od6/public/basic?alt=json-in-script"
        ex:converter="googleSpreadsheets" />

      <!-- include exhibit api -->
      <script src="http://static.simile.mit.edu/exhibit/api-2.0/exhibit-api.js"
type="text/javascript"></script>

      <style type="text/css">
        body { font-size: 0.8em; }
        #exhibit-wrapper { margin: 1em; border: 1px solid black; padding: 1em; }
      </style>

    </head>
    <body>
      <h1>Josef Dabernig - CV</h1>

      <!-- beginning of the area, where exhibit will do its magic -->
      <div id="exhibit-wrapper">

        <div ex:role="viewPanel">
            <div ex:role="view"></div>
        </div>

      <!-- end of the area, where exhibit will do its magic -->
      </div>
    </body>
  </html>
```

The first notable part of the code is `link to cv data`, realized by an HTML link to it. As stated before, the CV data is served by a Google Spreadsheet. Besides including data from Google Spreadsheets, Exhibit also supports a number of additional data sources, there exist various possibilities which are described at chapter 4.3 Data Options.

Next comes `include exhibit api` through linking to its freely available Javascript library.

Within the `exhibit-wrapper` one notes, that Exhibit introduces its own XML name space. The `ex:role="viewPanel"` attribute on the div-container tells Exhibit, that this container should act as a *ViewPanel*. The *ViewPanel* contains a single v*iew* specified by `ex:role="view"` in this very basic example.

4.1.3 Examining the first Result

The following screen shot shows the resulting, minimal *exhibit*:

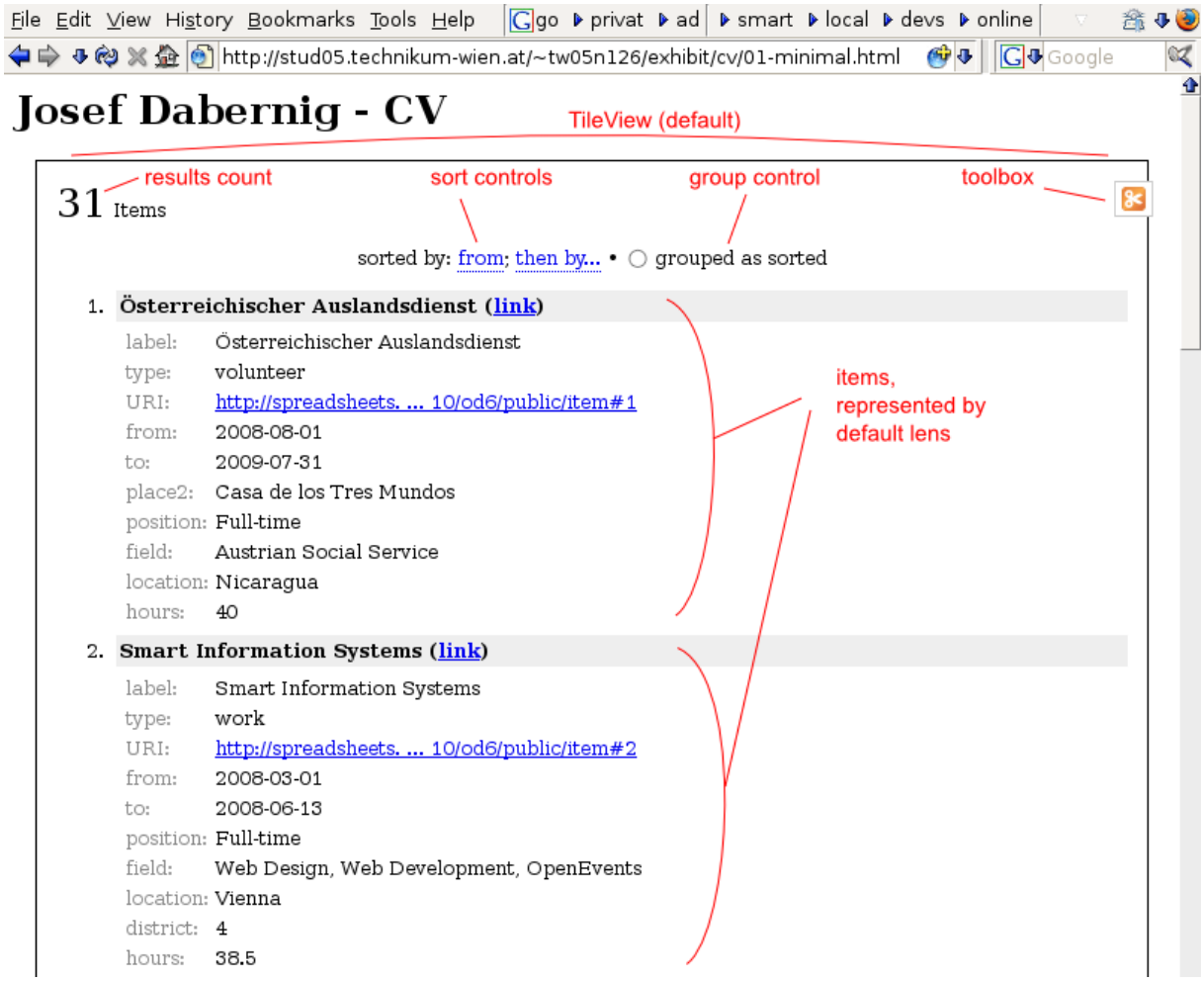


Illustration 4: Screenshot of a minimal exhibit

The black box around most of the content represents what we defined as `exhibit-wrapper` before. Inside, Exhibit created the following elements:

- a **default view** (TileView) containing all other Exhibit-specific elements:
- the **results count** represents the number of items displayed
- the **sort controls** allow to sort on the attributes specified in the spreadsheet
- the **grouping control** toggles grouping on the field, primarily sorted on
- the **toolbox** provides the data of the current *exhibit* in export formats for copy & paste
- a list of **items**, ordered as selected within the sort controls
- a **default lens**, represents each item and lists all attributes available for the item

To summarize, in order to create a minimal *exhibit*, one has to include the data, include the Exhibit API, define a *ViewPanel* and a *view*.

> The source code and the visual result of this step may be examined at
> http://stud05.technikum-wien.at/~tw05n126/exhibit/cv/01-minimal-setup.html

The following chapters provide information on using different data sources and how-to extend the minimal *exhibit* in functionality.

4.2 HTML Configuration Options

Exhibit offers a variety of HTML configuration options in order to configure and enhance elements of an *exhibit*. This chapter discusses the possible options, divided into the following sections:

- Include Exhibit API Extensions
- Add & Customize Views
- Add & Customize Facets
- Add & Customize Lenses
- Use Coders, Coordinators, Expressions and Formats

4.2.1 Include Exhibit API Extensions

Exhibit API extensions are packages of Javascript files, adding extra functionality to Exhibit. Two common extensions are the *Time Extension* and the *Map Extension*, which both add extra *view* types.

An Exhibit API extension is included by linking to it's main Javascript file. This step must not come before including the Exhibit API itself.

For example the *Time Extension* can be included, using the following HTML statement:

```
   <script src="http://static.simile.mit.edu/exhibit/extensions-2.0/time/time-
extension.js"></script>
```

The *Time Extension* will be used during the next chapter in order to extend the functionality for the personal CV example *exhibit*.

4.2.2 Add & Customize Views

In Exhibit, "*views are ways of looking at collections of items*" (DALE, 2008a). The minimal setup included a *view*, but didn't specify its type, therefore it was defaulted to a *Tile View*.

Reading the Exhibit for Authors wiki page (DALE, 2008b) leads to the insight, that Exhibit offers a variety of *views* besides the default *Tile View*. Exhibit supports out-of-the-box *Tile Views*, *Thumbnail Views* and *Tabular Views*. Other *views* require Exhibit API extensions, such as the above stated *Map View* and *Timeline View*, but also a *Timeplot View* might be added via an extension.

For the personal CV use case, the *Timeline View* and the *Tabular View* where considered to be the most appropriate. Luckily, both are well documented at the SIMILE wiki (CAMFIELD, 2008 and PFRED60, 2008). Following the instructions on the stated wiki pages, the minimal *exhibit* can be extended:

As the *Timeline View* requires the *Exhibit Time extension*, it will be included right after the *Exhibit API inclusion*:

```
  <!-- include exhibit time extension -->
     <script src="http://static.simile.mit.edu/exhibit/extensions-2.0/time/time-
extension.js" type="text/javascript"></script>
```

Adding a *Timeline View* similar to the creation of a default view, but additional parameters specify it more precisely:

```
  <div ex:role="view"
      ex:viewClass="Timeline"
      ex:start=".from"
      ex:end=".to"
      ex:topBandUnit="year"
      ex:topBandPixelsPerUnit="100"
      ex:bottomBandUnit="decade"
      ex:bottomBandPixelsPerUnit="500"
      ></div>
```

- the `ex:viewClass` attribute specifies the type of *view* to use
- `ex:start` and `ex:start` are required to define the time information
- the optional `ex:...Unit` statements adjust the zoom-factors of the *Timeline View*

(CAMFIELD, 2008)

The creation of the *Tabular View* works similar:

```
  <div ex:role="view"
      ex:viewClass="Tabular"
      ex:columns=".from, .to, .type, .label, .place, .position, .field"
      ></div>
```

- the `ex:viewClass` attribute specifies the type of view to use
- the `ex:columns` attribute specifies, which data to display

(PFRED60, 2008)

The examination of the results follows:

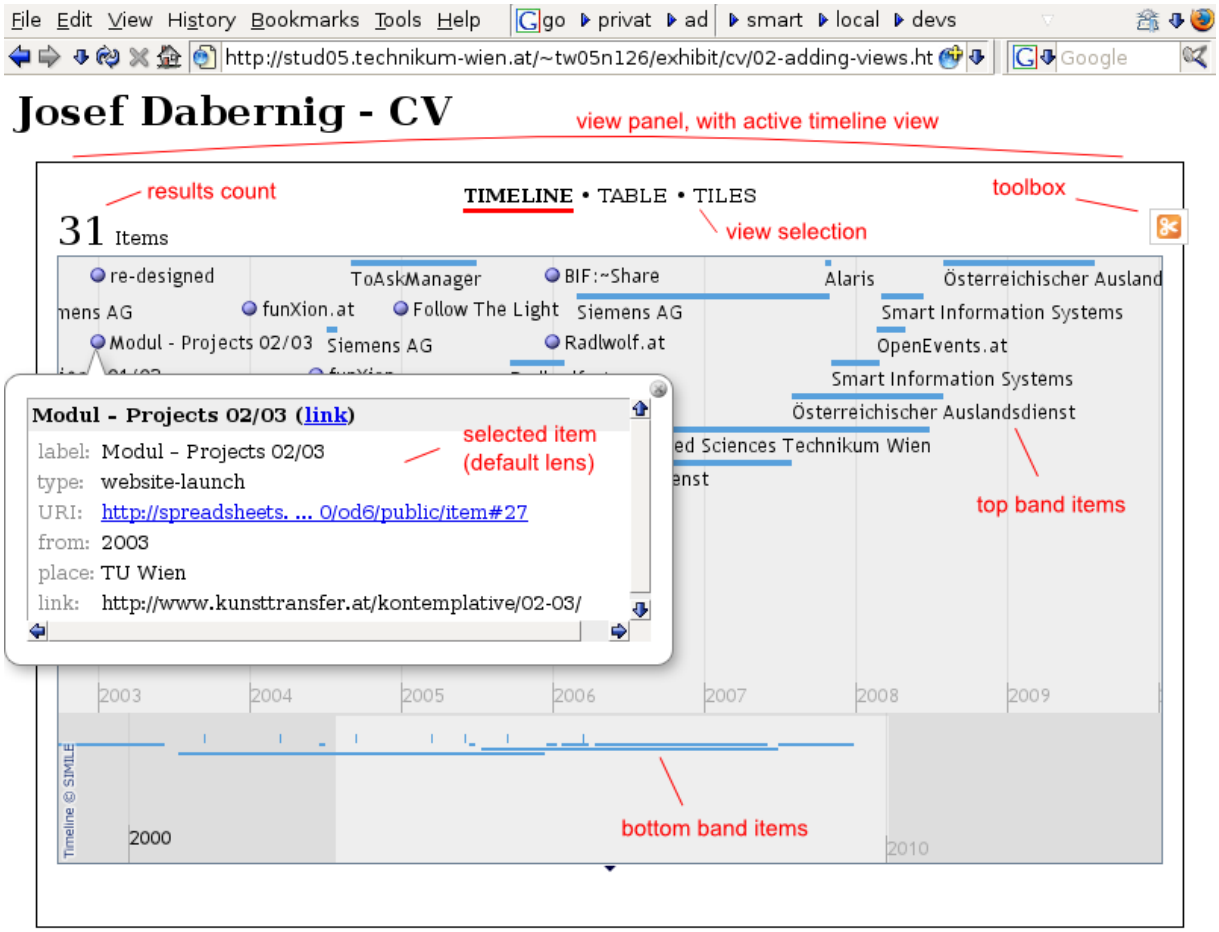
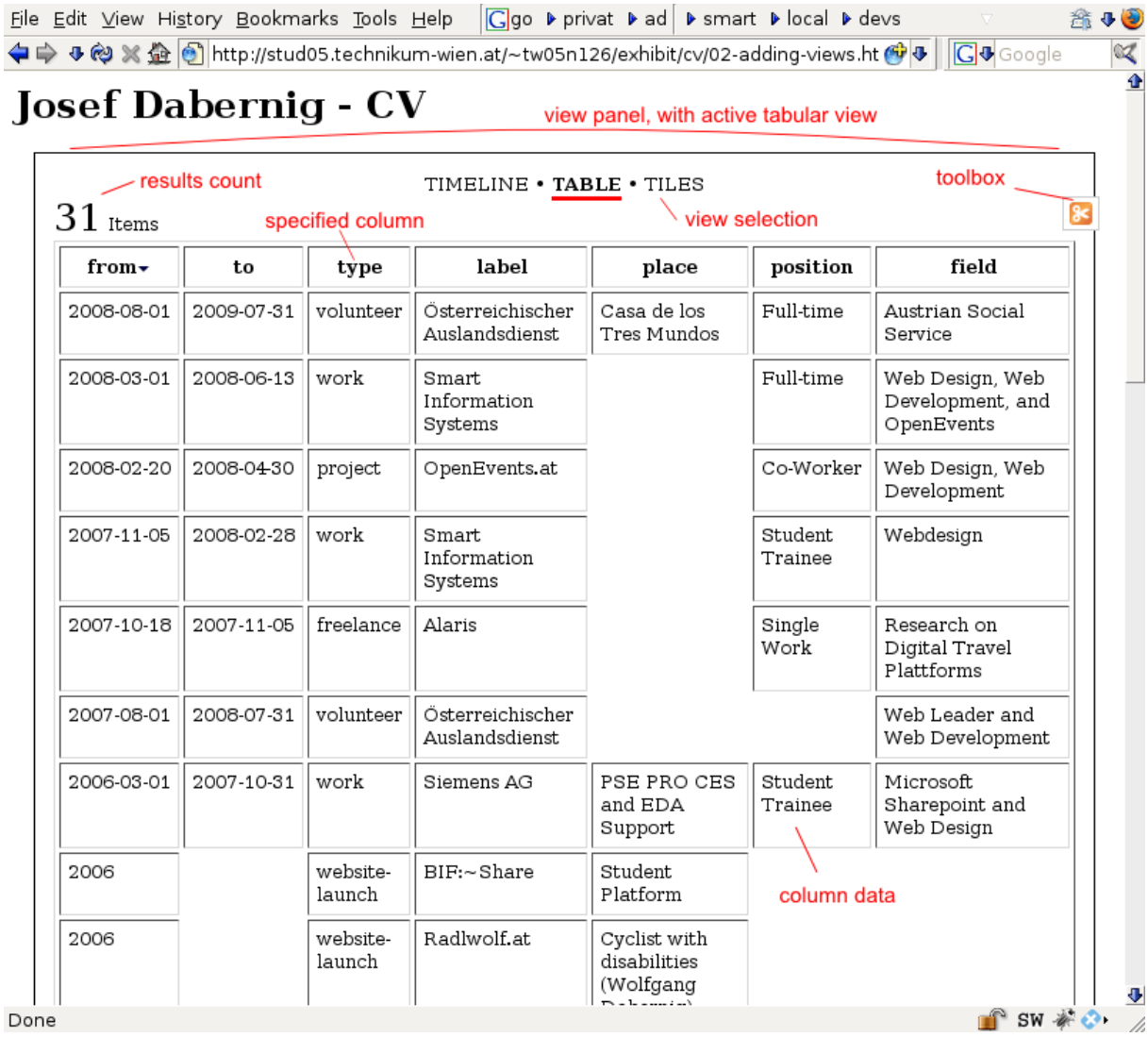


Illustration 5: Screenshot of an exhibit, containing multiple views and an active Timeline View

Besides already familiar elements like the **results count** and the **toolbox**, the following elements are new:

- a **view selection** allows to switch between the specified *views*
- the selected **Timeline View**, contains a **top band** and a **bottom band**, both showing the **items** provided by the spreadsheet on a time axis.
- the **selected item** is a pop up, Exhibit creates, when clicking on an item
- again, the selected item is represented by a **default lens**

Selecting the *Tabular View* from the **view selection** switches the *view* and produces the following screen shot:



from	to	type	label	place	position	field
2008-08-01	2009-07-31	volunteer	Österreichischer Auslandsdienst	Casa de los Tres Mundos	Full-time	Austrian Social Service
2008-03-01	2008-06-13	work	Smart Information Systems		Full-time	Web Design, Web Development, and OpenEvents
2008-02-20	2008-04-30	project	OpenEvents.at		Co-Worker	Web Design, Web Development
2007-11-05	2008-02-28	work	Smart Information Systems		Student Trainee	Webdesign
2007-10-18	2007-11-05	freelance	Alaris		Single Work	Research on Digital Travel Plattforms
2007-08-01	2008-07-31	volunteer	Österreichischer Auslandsdienst			Web Leader and Web Development
2006-03-01	2007-10-31	work	Siemens AG	PSE PRO CES and EDA Support	Student Trainee	Microsoft Sharepoint and Web Design
2006		website-launch	BIF:~Share	Student Platform		
2006		website-launch	Radlwolf.at	Cyclist with disabilities (Wolfgang		

Illustration 6: Screenshot of an exhibit, containing multiple views and an active Tabular View

Note, that the table generated by Exhibit contains sortable columns.

> The source code and the visual result of this step may be examined at
> http://stud05.technikum-wien.at/~tw05n126/exhibit/cv/02-adding-views.html

Further information on *views* is provided within David Huynh's thesis (HUYNH, 2007a, p.63).

4.2.3 Add & Customize Facets

"*A facet is a component whose purpose is to filter a collection's root set down to a filtered set*" (HUYNH, 2007a, p.62). The best way to understand *facets* is using them. After the following code definitions, a screen shot containing *facets* and further explanation is provided.

At the beginning of the `exhibit-wrapper div`, a `facets-wrapper div` is introduced:

```
  <div id="facets-wrapper">
  ...
  </div>
```

The `facets-wrapper` is formatted using the following CSS rule:

```
  #facets-wrapper { width: 15em; float: left; font-size: 0.9em; margin-right:
0.5em; }
```

Now, *facets* can be added to the `facets-wrapper`. The following code will produce a simple *Text Search Facet*, which allows the user to search over all attributes, available:

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

It would also be possible, to specify an *expression*, the facet should filter on. We define this and other settings for the following *List Facet*:

```
  <div ex:role="facet" ex:expression=".type" ex:facetLabel="Type"
ex:sortMode="value" ex:selection="work;study;volunteer;freelance"
ex:height="130"></div>
```

Note the following parameters:

- the `ex:role` attribute classifies the block as a *facet*
- no `ex:type` attribute was specified, therefore Exhibit will use *List Facet* as the default *facet* type
- the `ex:expression` attribute assigns the *List Facet* to filter on the type attribute of the CV data
- the `ex:facetLabel` attribute overrides the default *facet* label
- the `ex:sortMode` attribute overrides the default sortMode from count to value
- the `ex:selection` attribute provides an initial selection for the facet
- the `ex:height` attribute sets the facets height

Two more List Facets provide filtering on the two more attributes of the CV data:

```
  <div ex:role="facet" ex:expression=".label" ex:facetLabel="Place"
ex:sortMode="count"></div>

  <div ex:role="facet" ex:expression=".field" ex:facetLabel="Field"
ex:sortMode="count" ex:showMissing="false"></div>
```

The only new attribute introduced here is `ex:showMissing` – it will prevent the *facet* from adding an entry for items which do not provide any value for the expression, the *facet* filters on.

This is the resulting screen shot:

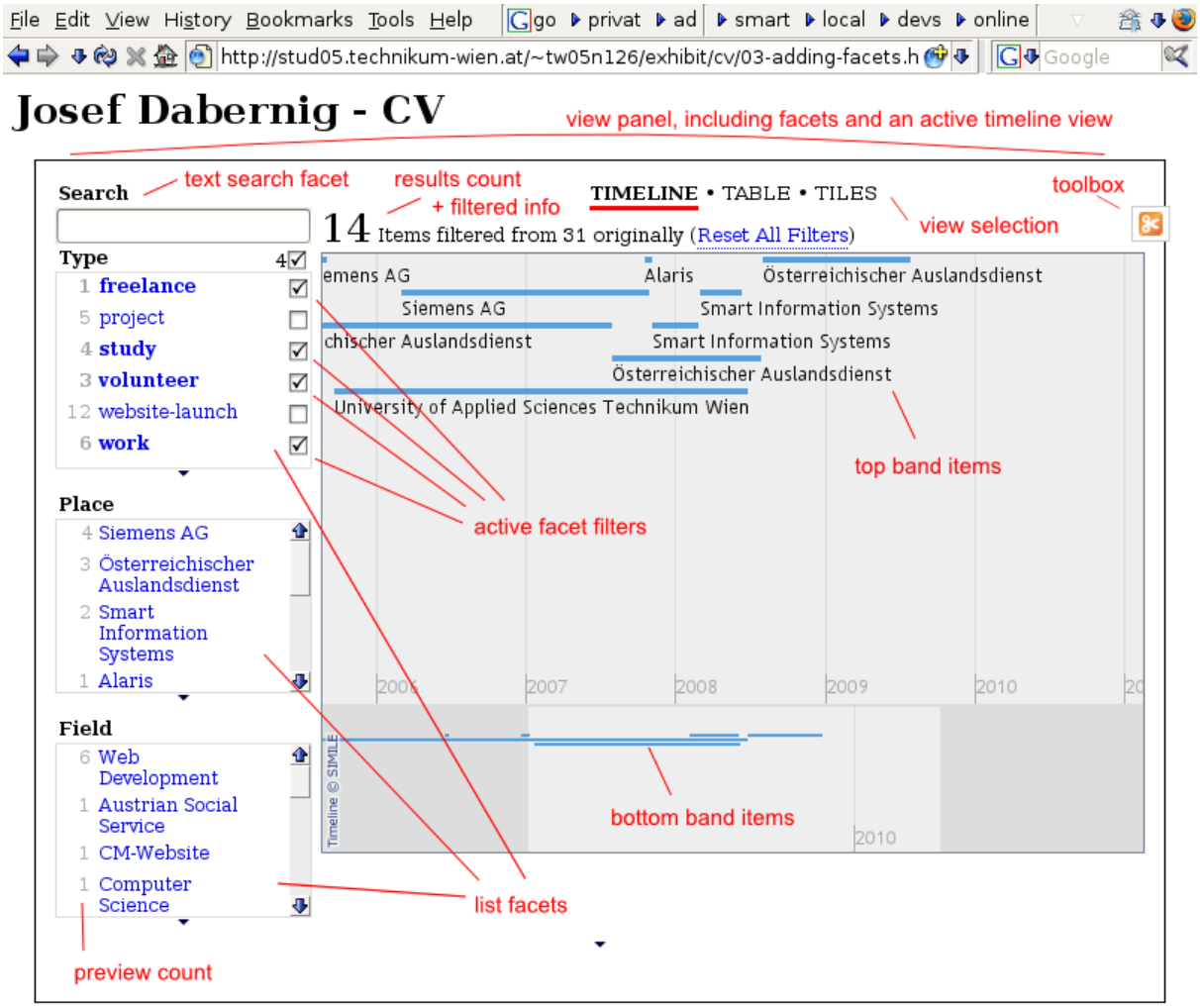


Illustration 7: Screenshot of an exhibit, containing multiple views and facets

The following elements are new:

- the **Text Search Facet** at the top of the left column provides full-text filtering functionality
- the **List Facet** named “Type” has four selected elements (these have been selected automatically by the selection parameter in the code)
- the **results count** additionally shows, that the items in the list have been **filtered** (due to the selection in the facets at the left)
- all *List Facet* items provide a **preview count**, which tells the user the number of items to expect when clicking on this particular item

> The source code and the visual result of this step may be examined at
> http://stud05.technikum-wien.at/~tw05n126/exhibit/cv/03-adding-facets.html

Further information on *facets* may be found in David Huynh's (2007a, p. 62) thesis.

4.2.4 Add & Customize Lenses

"*Lenses are ways of formatting individual items*" (DALE, 2008a). This means, that a lens may be used configure the visual appearance of single items of the personal CV.

Similar to the creation of *facets*, the element to wrap a *lens* receives the appropriate `ex:role` attribute:

```
<div ex:role="lens" style="display: none" class="lens-wrapper">
...
</div>
```

The `ex:content` attribute will insert the specified content into the element. Note that the content has to be empty, otherwise it doesn't work (LDD, 2007).

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

HTML attributes may be generated, using the `ex:*-content` attribute. In this example `ex:href-content` sets the `href` attribute for the HTML link to the item's `link` property:

```
<div class="label"><a ex:href-content=".link" ex:content=".label"></a></div>
```

Control logic might be added using `ex:if-exists`, `ex:if` or `ex:select` elements. The following code snipped adds the hours of work to the time information only if it exists, using the `ex:if-exists` attribute:

```
<div class="time">
    from <span ex:content=".from"></span>
    to <span ex:content=".to"></span>
    <span class="hours" ex:if-exists=".hours">
      (<span ex:content=".hours"></span> hours)</span>
</div>
```

An example of embedding function within the *expressions*-aware `ex:content` attribute:

```
<div class="place" ex:content="concat('place: ',.place)"></div>
```

Two further lines of code add extra information to the *lens* template:

```
<div class="field" ex:content=".field"></div>
<div ex:if-exists=".location" ex:content=".location"></div>
```

Additional CSS code adjusts the representation of the *lenses*:

```
.lens-wrapper { width: 25em; }
.lens-wrapper .type { float: right; color: #888; }
.lens-wrapper .time { font-style: italic; font-size: 0.8em; }
```

The added *lens* produces the following screen shot, when an *item* has been clicked:

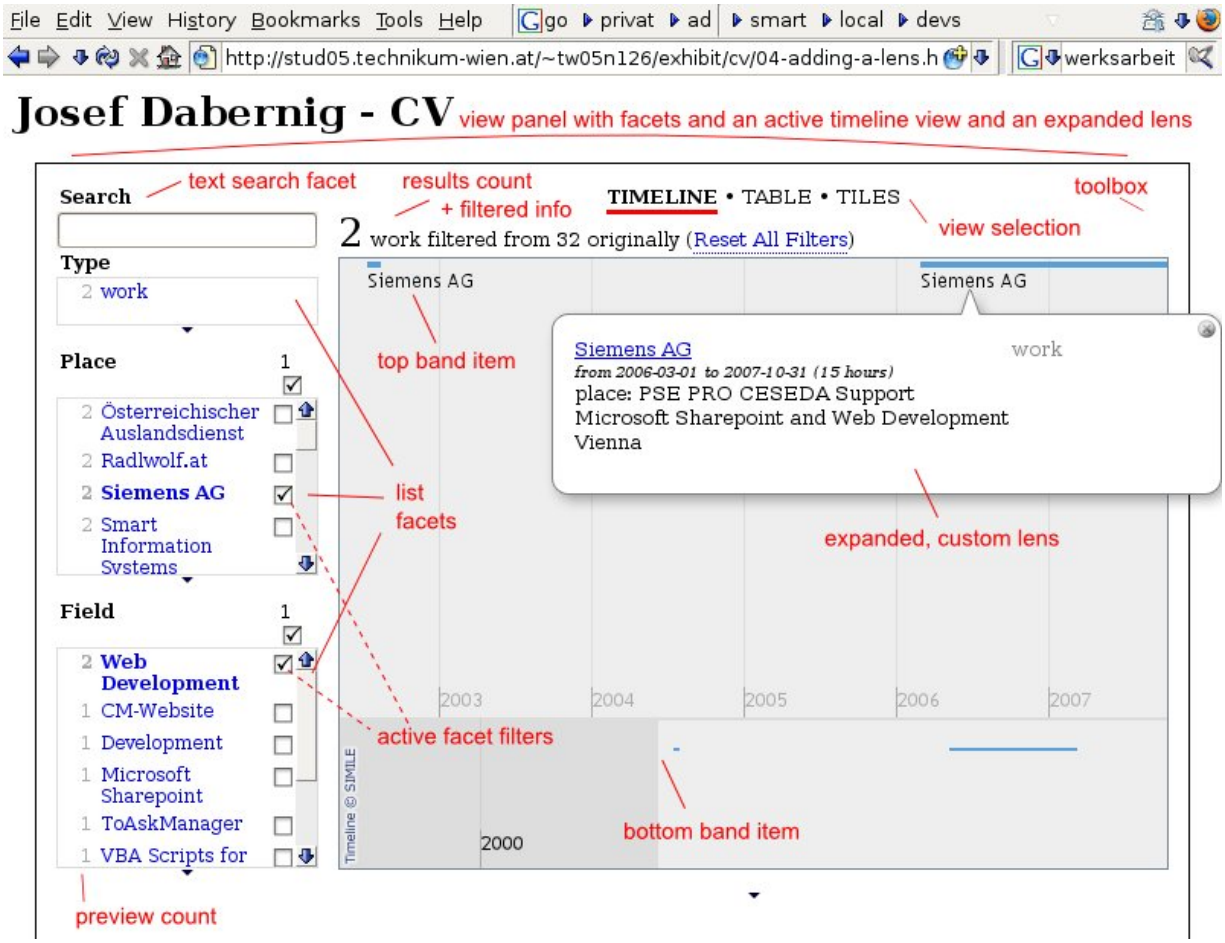


Illustration 8: Screenshot of an exhibit, containing multiple views, facets and lenses

Notice the expanded, custom *lens*. In comparison to the default *lens*, first seen in 4.1.3 Examining the first Result, this custom *lens* doesn't simply list all attributes, available for the specific item. It displays the item in a the way, the *lens template* in the code specifies.

It is also possible, to add item type-specific lens templates, using the `ex:itemTypes` attribute:

```
  <!-- website-launch specific lens -->
  <div ex:role="lens" style="display: none" class="lens-wrapper"
ex:itemTypes="website-launch">
    <div class="type" ex:content=".type"></div>
    <div class="label"><a ex:href-subcontent="{{.link}}?from=CV"
ex:content=".label"></a></div>
    <div class="time">launched <span ex:content=".from"></span></div>
    <div class="place" ex:content="concat('for: ',.place)"></div>
  </div>
```

This defines a custom *lens* template for items having the type `website-launch` only. Items of all other types will use the more generic one without the `ex:itemTypes` parameter, if available or otherwise a default *lens*.

Also note the `ex:href-subcontent` attribute. Its advantage over `ex:href-content` is the possibility to create dynamic URLs because it may contain *expressions* which have to be encapsulated using a `{{ }}` syntax (KARGER, 2007).

> The source code and the visual result of this step may be examined at
> http://stud05.technikum-wien.at/~tw05n126/exhibit/cv/04-adding-lenses.html

Further information on *lenses* is available in David Huynh's (2007a, p.63) thesis.

4.2.5 Use Coders, Coordinators, Expressions and Formats

Besides the stated methods of customizing an *exhibit* by adding and customizing *views*, *facets* and *lenses*, Exhibit offers even more features:

- *coders* "*translate a piece of information to some visual feature*". (ANDRISI, 2008) More information: http://simile.mit.edu/wiki/Exhibit/2.0/Coders
- *coordinators* "*synchronize selection and highlight among several views*". (HUYNH, 2007b) More information: http://simile.mit.edu/wiki/Exhibit/2.0/Coordinators
- *expressions* "*make it convenient to access its [Exhibit's] graph-based data model*" . (HUYNH, 2008a, p.58) More information: http://simile.mit.edu/wiki/Exhibit/Expressions
- *formats "modify the ways values of various value types (date, url, image, etc.) are rendered*". (SABERHAGEN427, 2008) More information: http://simile.mit.edu/wiki/Exhibit/2.0/Formats

Their explanation goes beyond the scope of thesis, but the interested reader is encouraged to refer to chapter 5.1 Sources of Information.

4.3 Data Options

The article "Creating, Importing, and Managing Data" at Exhibit's SIMILE wiki section states:

"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". (DASJO, 2008)

The following listing summarizes the possibilities stated and explained in detail at the wiki article:

- Manual Creation and Management of JSON Data
- Manual Conversion to JSON using Babel
  - BibText, Excel, JPEG, N3, RDF/XML, Tab-Separated Values
- Live Conversion to JSON using Importers
  - Babel-based Importer (BibTex, Excel, RDF/XML & N3)
  - Google Spreadsheet Importer
  - RDFa Importer

It is notable, that as all features of Exhibit, importers are free to extend. For example the *RDFa Importer* was contributed by an Exhibit user named Keith Alexander.

(DASJO, 2008)

5 Technical Documentation of Exhibit

The technical documentation of Exhibit includes a listing of sources of information, available on Exhibit. It further discusses the topic API documentation and presents the two prototypes “Exhibit Code Documentation” and “Exhibit SVN History”, both realized using Exhibit.

5.1 Sources of Information

Besides the information, provided within this thesis, these sources provide developers with useful documentation on Exhibit:

- The **official web site of Exhibit** is http://simile.mit.edu/exhibit/
- The **Exhibit wiki** is located at http://simile.mit.edu/wiki/Exhibit and open to contributions. It contains numerous tutorials on the usage of Exhibit, but also gives technical background information.
- The **SIMILE Widgets Google Group** is the official discussion forum for Exhibit-related questions and located at http://groups.google.com/group/simile-widgets. David Huynh, Exhibit's core developer regularly answers questions in this group.
- The **MIT Simile Web Widgets Google Code Project** hosts Exhibit's source code and may be accessed via http://code.google.com/p/simile-widgets/. Note, that at the time of research, documentation on Exhibit hasn't been moved over to the Google Code project from the Exhibit wiki at SIMILE.
- The **Exhibit API** is available remotely as a free service via http://static.simile.mit.edu/exhibit/, where both stable and development versions are provided.
- The former Exhibit repository, located at http://simile.mit.edu/repository/exhibit is outdated and has been moved to the **MIT Simile Web Widgets Google Code Project**.

The following publications on Exhibit are available:

- **User Interfaces Supporting Casual Data-Centric Interactions on the Web** is David Huynh's doctoral thesis providing in-depth information on Exhibit and available from http://davidhuynh.net/media/thesis/thesis.php
- **Exhibit: Lightweight Structured Data Publishing** was written by David Huynh, Robert Miller and David Karger and is the first paper, published on Exhibit. It is available on http://davidhuynh.net/media/papers/2007/www2007-exhibit.pdf

Example *exhibits* can be found at the following web addresses:

- The official web site of Exhibit: http://simile.mit.edu/exhibit/
- The examples page of the Exhibit wiki: http://simile.mit.edu/wiki/Exhibit/Examples
- The example directory of the Exhibit source code: http://simile-widgets.googlecode.com/svn/exhibit/trunk/src/webapp/site/examples/
- David Huynh's project folder http://people.csail.mit.edu/dfhuynh/projects/

5.2 API Documentation

Popular JavaScript frameworks like Prototype and Dojo provide API documentation (Prototype Core Team, 2008 and The Dojo Foundation, 2008). Research turned out, that a documentation generator for JavaScript exists – JsDoc toolkit (MICMATH, 2008). The main purpose of JsDoc toolkit is to generate documentation from parsing JavaScript files for inline comments. This is comparable to what the more popular Javadoc tool does for Java source code (Sun Microsystems, Inc., 2008).

Currently, most of the source code files of Exhibit don't contain any or contain very little documentation. A test-run of the JsDoc toolkit on Exhibit's source code[1] produced an unusable result of very low quality. The toolkit recognized hardly any of the existent classes, which Exhibit consists of.

5.3 “Exhibit Code Documentation” *exhibit*

To accomplish the target “Extend documentation, available on Exhibit”, an effort on creating an “Exhibit Code Documentation” has been made. As chapter 5.2 API Documentation explains, at the moment Exhibit's code is fairly documented and there doesn't exist a proven method to generate documentation from the code itself.

In order to propose a possible documentation, a prototype was created using Exhibit on its own: the “Exhibit Code Documentation” *exhibit*. Its target is to provide developers a reference on the basic structure of Exhibit's code base. For this purpose, a Google Spreadsheet was created. The spreadsheet contains a row for every folder and file, the Exhibit source code consists of. Exhibit interprets each row as an item. Items are described by their attributes, which are the corresponding columns in the Spreadsheet.

The following columns where specified:

- The **type** column defines the type of the item
  - 3.3 Architecture provides an overview of the components, Exhibit consists of. These components where matched to the types.
  - The type “*component*” refers to a folder, containing multiple components. For example the UI component contains sub-components like *view*, *facet* and *lens*.
  - The type “*extension*” refers to Exhibit extensions, as described in chapter 4.2.1 Include Exhibit API Extensions.
  - The type “folder” refers to a folder of less significance, compared to folders of the *component* type. For example the *Map Extension* contains a folder named “scripts”, where the *Map View* is defined. The “scripts” folder is just a structural folder, but not a *component*.
- The **id** column provides a unique identifier for the item
  - The scripts of Exhibit's source code are accessible via “http://simile-widgets.googlecode.com/svn/exhibit/trunk/src/webapp/api/scripts”. The id is the relative path below this scripts folder. For example, the UI component is located at http://simile-widgets.googlecode.com/svn/exhibit/trunk/src/webapp/api/scripts/ui, so its id is “/ui”.
  - *Extensions* are located within a different folder, than the Exhibit's “core” source code: http://simile-widgets.googlecode.com/svn/exhibit/trunk/src/webapp/extensions/. This prevents them from being mapped using the relative id approach stated before. Two options where considered, to deal with this issue: 1) Raise the base folder for calculating the ids up to http://simile-

1 http://simile-widgets.googlecode.com/svn/exhibit/trunk/src, version 1383

widgets.googlecode.com/svn/exhibit/trunk/src/webapp/. This means consistent ids for all items, but means longer ids. 2) Map *Extensions* to the virtual base id "/extensions". While this approach keeps the ids short, it's draw back is, that ids cannot be consistently mapped to their full path. For example, the *Timeline Extension* would be mapped from "/extensions/time" to the following, invalid path: http://simile-widgets.googlecode.com/svn/exhibit/trunk/src/webapp/api/scripts/extensions/time. The second option, the implementation of a separate id-mapping for *Extension*-related items was chosen.

- The **parent** column refers to the id of the items parent item.
- The **label** column is a human-readable representer for the item. In general, this refers to the class name specified within a *component*'s source, without the "Exhibit" name space. For example, the source file of the *ColorCoder*, located at http://simile-widgets.googlecode.com/svn/exhibit/trunk/src/webapp/api/scripts/ui/coders/color-coder.js, states "Exhibit.*ColorCoder*" in its initial source comment.
- The **description** column contains a description of the item's purpose
- The **uri** column refers to SVN path to the item's source code. The generation of the uri is described in the explanation of the id column above.
- The **wiki-link** column contains zero, one or multiple references to pages at the Exhibit wiki which contain documentation on the item.

The combination of information provided within David Huynh's thesis and study both of Exhibit's source code and the wiki, all user interface related folders and files where transferred to the Google Spreadsheet and documented.

Next, an *exhibit* based on the Google Spreadsheet was created, following the procedures described in chapter 4 Usage of Exhibit.

A separate JSON-file configures the non-text valueTypes of the columns, described above, to ensure that Exhibit will handle them properly:

```
 {
       properties: {
           "parent": {
               valueType:    "item"
           },
           "uri": {
               valueType:    "url"
           },
           "wiki-link": {
               valueType:    "url"
           }
       }
}
```

The following screen shot shows the visual appearance of the created “Exhibit Code Documentation” *exhibit:*

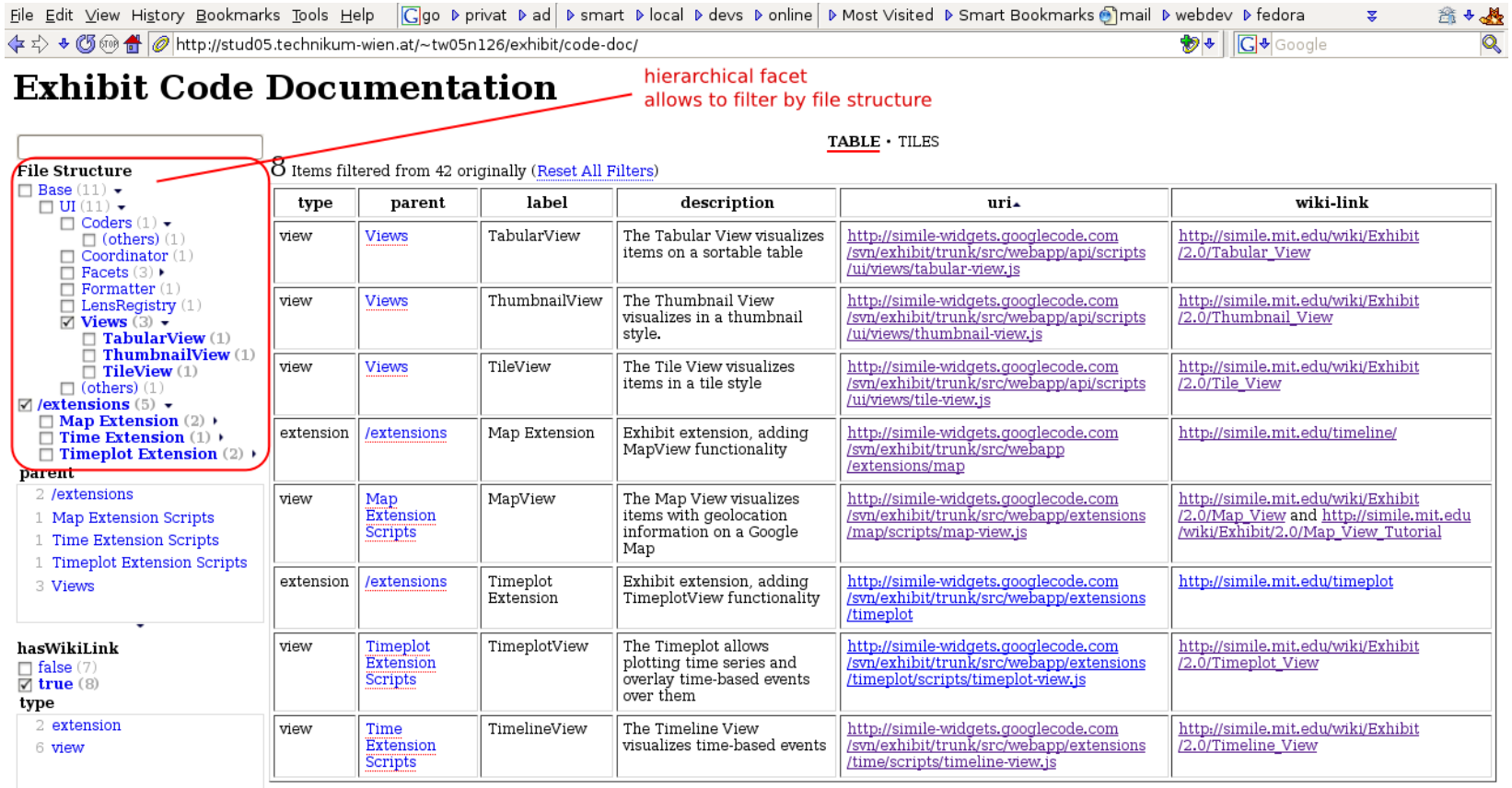


Illustration 9: Screenshot of the "Exhibit Code Documentation" exhibit

The reader may find one page element, not described before: The *facet* at the left, with the title “File Structure”, is a *Hierarchical Facet*. It allows to browse hierarchical data, which is provided by the **parent** column of the Google Spreadsheet, as described before.

To summarize, the “Exhibit Code Documentation” *exhibit* allows a user to interactively browse the Exhibit's source code, supported by the facets which allow to filter on file structure hierarchy and meta-data like the item types. It also provides access to further information as an items source code at the SVN repository and, if existent, information at the Exhibit wiki.

> The source code and the visual result of the “Exhibit Code Documentation” *exhibit* may be examined at http://stud05.technikum-wien.at/~tw05n126/exhibit/code-doc/

5.4 “Exhibit SVN History” *exhibit*

Comparable to the “Exhibit Code Documentation” exhibit, described previously in chapter 5.3, while writing this thesis, an *exhibit* was created in order visualize Exhibit's SVN activity. The aim of the “Exhibit SVN History” *exhibit* is to visualize activity on Exhibit's source code by transforming SVN log data to a Exhibit *Timeline View*. Using Exhibit to present the data allows advanced features like *facets* for filtering.

The following Subversion command was executed on the Exhibit source in order to generate an export of Exhibit's SVN log in XML format:

```
svn log -v --xml > log.xml
```

The resulting log.xml file contains a large number of `logentry` items. One of them is listed in the following example:

```
<logentry revision="1209">
    <author>dfhuynh</author>
    <date>2007-11-19T14:05:38.214615Z</date>
    <paths>
        <path action="D">/exhibit/branches/2.0</path>
        <path copyfrom-path="/exhibit/branches/2.0" copyfrom-rev="1208"
action="A">/exhibit/trunk</path>
    </paths>
    <msg>
        Moving branches/2.0 to trunk for real this time, hopefully.
    </msg>
</logentry>
```

The semantics of the data are sufficient to create a meaningful *exhibit* from it. The `date` attribute of every `logentry` allows to display SVN activity within a Timeline View. The `author` attribute provides information on who committed the changes and the `paths` list allows to understand, which files have been modified. The `msg` contains the comment, the author provided when committing the change.

Transforming the Exhibit SVN log data, so that Exhibit would be able to read it, was the most challenging task. As described previously in chapter 4.3 Data Options, several *importers* exist to import various file formats. Unfortunately, at the time of research, there didn't exist an importer which could import the xml data, describes above. The *RDF/XML Importer* doesn't work in this case, because the xml data doesn't contain name space declarations, which are required by the *importer*. Writing a custom importer of course was an option, but out of scope of this thesis. As a workaround, the XML data was transformed to a spreadsheet, using Microsoft Excel. The spreadsheet data then was copied to a Google Spreadsheet, although the performance of Google Docs was weak, due to the large amount of data.

The imported data contains all SVN changes related to Exhibit up to revision 1360, but the total number of items is 4002. This is due to a flattening process, Microsoft Excel applies when important hierarchical XML data. For example the above stated example contains a single `logentry`, but when imported two rows are generated, because it contains two `path` nodes.

The “Exhibit SVN History” *exhibit* produces the following visual output:

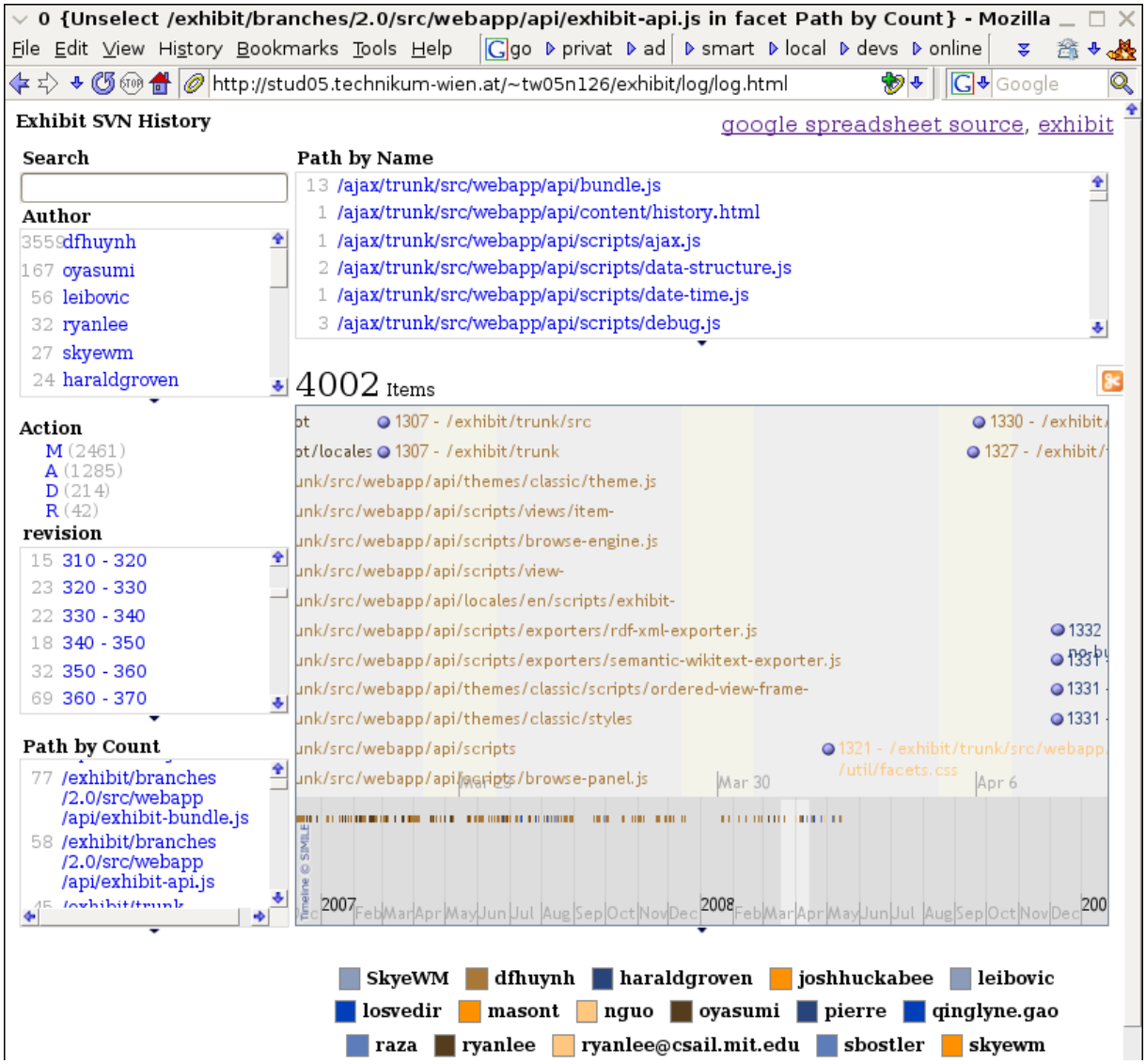


Illustration 10: Screenshot of the "Exhibit SVN History" exhibit

Note, that besides already familiar *facets* and the *Timeline View*, this *exhibit* displays its items in a “color coded” way. The *Timeline View* configuration contains a setting, which forces the *view* to display items colored according to their **author** attribute and also generates a legend, visible at the screen shots bottom. This is the responsible configuration:

```
ex:colorKey=".author"
```

The “Exhibit SVN History” *exhibit* provides the interested user with a powerful tool which helps him analyze activities on Exhibit's source code.

6 Real-World Use Cases based on Exhibit

This chapter shows practical use cases based on Exhibit.

During the writing process of this thesis, two *exhibits* have been created, which are both described within chapter 5 Technical Documentation of Exhibit.

Besides the already discussed *exhibits*, the following use cases show that it is possible, to use Exhibit in dynamic web applications:

- **Auslandsdienst WebNeu** is a dynamic web application based on Plone, an open source content management system (Plone Foundation, 2008)
- **OpenEvents** is a dynamic web application developed with Java Servlet and JSP technologies

6.1 Auslandsdienst WebNeu

"Auslandsdienst" refers to german name of the "*Austrian Service Abroad", a non-profit initiative, founded 1998 by Andreas Maislinger. The organization provides positions for an alternative Austrian national service all over the world*. (Wikipedia, 2008).

"WebNeu" is a project, initiated by members of the Austrian Service Abroad in order to create a dynamic web application which will replace the current, static web site of the organization, but hasn't been released, yet (Österreichischer Auslandsdienst, 2008). WebNeu is built upon Plone which is based on the open source application server Zope (Zope Corporation, 2008). Exhibit is used in order to visualize data on a Google Map or in a *Timeline View*. This chapter explains, how WebNeu's *exhibits* have been realized.

A way to display content in Plone since version 3 are *Zope 3 browser views*. Using this approach, a *View Class* provides data which a *Template* uses to generate output. (ASPELI, 2008)

> More information on Zope 3 Browser views can be found at
> http://plone.org/documentation/tutorial/customization-for-developers/zope-3-browser-views

The following Zope Configuration Markup Language (ZCML – RICHTER, 2008) declaration defines a *browser view*:

```
    <browser:page
       for="Products.AuslandsdienstBase.content.interfaces.IOrganisation"
       name="organisation-view"
       class="Products.AuslandsdienstBase.browser.OrganisationView.OrganisationVi
ew"
       template="templates/organisation-view.pt"
       />
```

- The `for` attribute assigns the browser view to objects of the type Organisation
- The `name` attribute defines the path at which the browser view will be accessible
- The `class` attribute specifies the *View Class* which provides data for the template
- The `template` attribute defines the *Template* which will be used to render the object

6.1.1 Provide data using View Classes

WebNeu's *View Classes* where extended to provide data in JSON format for Exhibit. This was accomplished by using *simplejson*, a JSON encoder/decoder for Python (IPPOLITO, 2008a). s*implejson* is able to encode basic Python object hierarchies out of the box. Creating an extended JSONEncoder allows to encode complex Python object hierarchies, containing custom types (IPPOLITO, 2008b). An AuslandsdienstEncoder was written in order to encode WebNeu-specific Python objects.

The following code represents the definition of the AuslandsdienstEncoder for the Organisation type, which defines an organizational unit within the Austrian Service Abroad:

```
class AuslandsdienstEncoder(simplejson.JSONEncoder):
    def default(self, obj):

        # Organisation type-specific encoding
        if isinstance(obj, Organisation):
            return {"label" : obj.Title(),
                    "id" : obj.getId(),
                    "type" : "Organisation",
                    "groupMembers" : obj.getGroupMembers(),
                    "latitude" : obj.getGeoLocation()[0],
                    "longitude" : obj.getGeoLocation()[1],
                    "geoLocation" : obj.getGeoLocation()  }

        # ... other WebNeu type-specific specific encodings ...

        return simplejson.JSONEncoder.default(self, obj)
```

The **AuslandsdienstEncoder** checks by using the `isinstance` method, if `obj` (the object to encode) is of a known type as the `Organisation` type, specified in this shortened example. If the type of the object matches, the type-specific encoding function `return`s a Python tuple containing the information to encode. *simplejson* is able to encode the result, because it is a basic Python object hierarchy. The same functionality has been implemented for various other WebNeu-specific content types.

A *View Class* function **exhibitJSON** was written to convert a list of objects into Exhibit's JSON format:

```
    def exhibitJSON(self, items):
        """
        serializes item tuples to JSON using simplejson
        JSON structure:
http://simile.mit.edu/wiki/Exhibit/Template/JSON_Data_File
        """
        tuple = {"items" : items}
        return simplejson.dumps(tuple, cls=AuslandsdienstEncoder)
```

First, the `items` list of objects to encode for Exhibit is packed into a `tuple` with a key named “items” in order to conform with the JSON Data File Template (HUYNH, 2007c). The function returns the JSON encoding of the `tuple` using the `simplejson.dumps` function. The optional `cls` parameter forces the function to use the custom `AuslandsdienstEncoder`, specified before.

At this point of time, it is possible to provide data in a way, Exhibit can parse.

6.1.2 Integrate Exhibit using the InlineImporter

Normally, Exhibit loads its data from an URL. In this case it was preferred to render the JSON code into the HTML page itself. Brian Rossmaier (2007) found a way to realize this using an *InlineImporter*. The *InlineImporter* allows to load data directly into Exhibit by using several JavaScript statements. An additional method for the Organisation View Class “exhibitImport” was created:

```
    def exhibitImport(self):
        """
        returns javascript command for script tag to load exhibitJSON string
        """
        data = self.exhibitJSON(self.subOrganisations() +
                                self.positions() +
                                self.jobs() +
                                [job.getServant() for job in self.jobs()])
        return "Exhibit.InlineImporter.userdata=(" + data + ");";
```

The function **exhibitImport** uses the previously defined **exhibitJSON** function to encode a list of all relevant objects for the Organisation into the `data` variable. For example, all sub organisations are accessed by the `self.subOrganisations()` getter. The function `return`s a JavaScript command containing the encoded JSON string which will be used in template, discussed in the following chapter.

6.1.3 Generate HTML Code using Templates

As specified in the Zope Configuration Markup Language (ZCML) declaration for the browser view at the beginning of this chapter, the browser view has been associated with a *Template* which will render the Organisation object into HTML code.

To set up Exhibit and the *InlineImporter*, a METAL macro block for the JavaScript-related fill-slot is defined (DE VITIS, 2008). METAL is a way to define macros for Zope Templates (SIMON, 2008).

```
<metal:block fill-slot="javascript_head_slot">
...
</metal:block>
```

Plone renders code within this block into the HTML head. First, the Exhibit API and the Time and the Map Extension are referenced, similar to the procedure described in chapter 4.1.2 Creating the Presentation. The following line tells Exhibit, that an *InlineImporter* will be used:

```
  <!-- load exhibit data from inline javascript -->
  <link type="inline" rel="exhibit/data" />
```

Further, several lines of JavaScript code initialize the *InlineImporter*:

```
  <script>
    Exhibit.InlineImporter = { };
    Exhibit.importers["inline"] = Exhibit.InlineImporter;
    Exhibit.InlineImporter.load = function(link, database, cont) {
      Exhibit.UI.showBusyIndicator();
      database.loadData(Exhibit.InlineImporter.userdata);
      Exhibit.UI.hideBusyIndicator();
      if (cont) cont();
    };
  </script>
```

This is the most important statement:

```
    <!-- exhibit inline data load operation including json data string comes from
view class -->
    <script tal:content="view/exhibitImport">
      Exhibit.InlineImporter.userdata=(JSONSTRING);
    </script>
```

Using the Template Attribute Language Expression Syntax (TALES) `tal:content` directive , the contents of the `script` tag will be replaced with the result of the exhibitImport function of the *View Class*. TALES is an expression language, used to write Templates for Plone and Zope (EVAN, 2008).

> More information on Templates can be found at
> http://plone.org/documentation/tutorial/zpt

6.2 OpenEvents

Target of the project OpenEvents is to develop an open [...] platform for registration and search of events [...]. The project OpenEvents is developed as a part of the ebSemantics project and aims to cross-link event data in the semantic-web-based e-commerce. [...] The project is supported by the Internet Privatstiftung Austria.

(Smart Information Systems, 2008)

Together with Svetlana Hollerer and Markus Klausz, fundamental parts of the OpenEvents platform have been developed at Smart Information Systems. As defined in the specification, the platform is realized using Java Servlet technologies in conjunction with Java Server Pages (JSP) and implements the Model-View-Controller (MVC) pattern.

- *Model – A data interface allows to access event data by providing Java Beans*
- *View – JSPs generate HTML code for the user's browser*
- *Controller – Java Servlets manage the web application's control flow*

(HOLLERER et al, 2008)

OpenEvents uses Exhibit to visualize event data. The integration of the Exhibit framework into OpenEvents was realized by using *InlineImporter* approach. It has already been discussed for the Auslandsdienst WebNeu project (see 6.1.2 Integrate Exhibit using the InlineImporter).

The following screen shot shows the main page of the OpenEvents platform, containing Exhibit elements like a *Maps View* and a *Timeline View*.

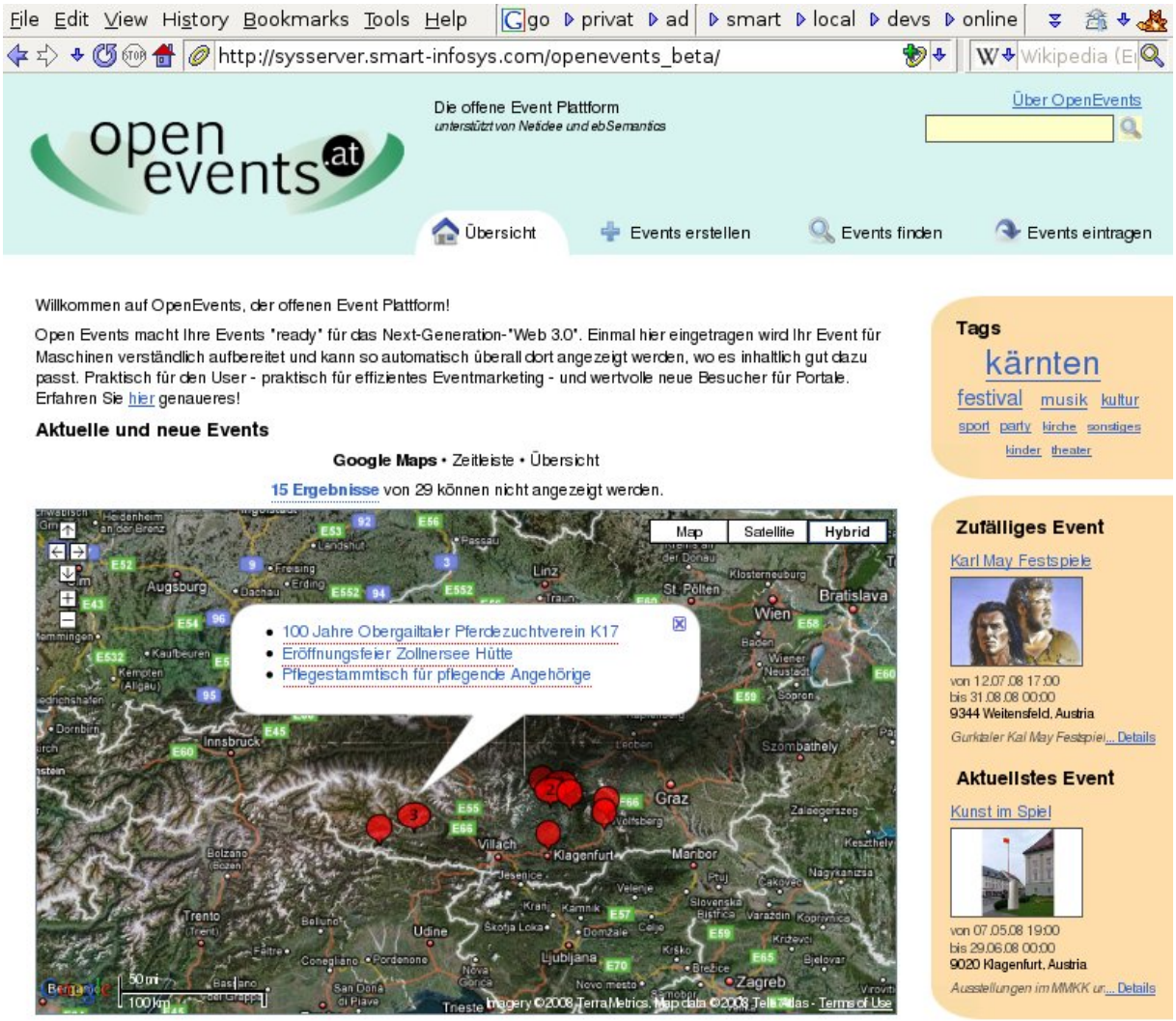


Illustration 11: Screenshot of the main page of the OpenEvents plattform

In comparison to the Auslandsdienst WebNeu project, which uses Python code, OpenEvents is developed using Java technologies. To transform event data, encapsulated by Java Beans into Exhibit's preferred JSON format, the *jsonmarshaller* (PEREZ, 2008a) was used. It allows to control the encoding by applying *jsonmarshaller*-specific annotations on Java Classes and their variables or getters (PEREZ, 2008b).

The search page of the OpenEvents platform heavily uses Exhibit:

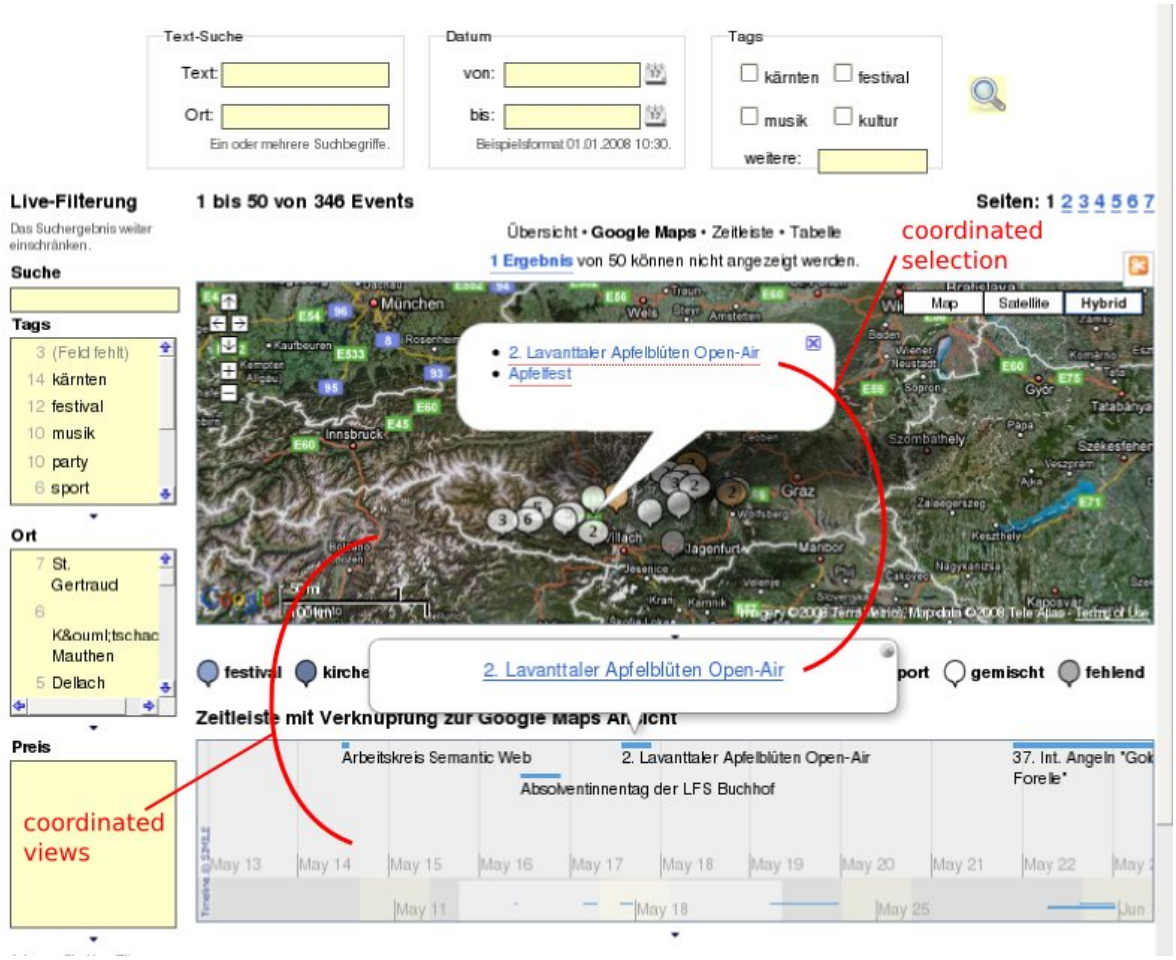


Illustration 12: Screenshot of the search page of the OpenEvents plattform

This page contains a new Exhibit feature, a *coordinator*. Its purpose is to synchronize selection (see the **coordinated selection** mark) and highlight among several views (see the **coordinated views** mark) (HUYNH, 2007d). Illustration 11 contains two *views*, a *Map View* and a *Timeline View*. To coordinate the two views, the following *coordinator* is defined:

```
<div ex:role="coordinator" id="event"></div>
```

Each view is configured to use the `event` coordinator using the `ex:selectCoordinator` attribute:

```
ex:selectCoordinator="event"
```

Clicking on an event in one of the both *views* doesn't only display the lens for the clicked event within the current *view*, but will also trigger the *event coordinator*. The coordinator forces the other *view* to display the event and its *lens* within the *view*.

In the example screen shot of Illustration 11, both *views* display the same event.

7 Conclusion & Discussion

“Creating interactive web apges using the Exhibit Framework” introduced Exhibit, a JavaScript framework enabling publishers to create data-centric and interactive web pages.

A detailed tutorial showed how-to create an interactive personal CV step-by-step. Interested users may follow the example to create for example their own personal CV or any other data-driven *exhibit*. Every single step of the creation process is documented and also refers to further documentation, available on the web.

References to existing technical documentation on the Exhibit framework have been provided and two real-world use cases demonstrated, that Exhibit also works well in database-driven three-tier web applications. Due to the limited scope of this thesis, some advanced components of the Exhibit framework could not be discussed in detail.

In the sequent, concluding statements, I'd like to analyze Exhibit's community, documentation and performance aspects and provide suggestions for the future.

7.1 Community

David Huynh's efforts in supporting the community by continuously answering questions and providing solutions and fixes to found bugs are outstanding. He nearly answers any question, asked on the newsgroup, which keeps the users' motivation up and also motivates others to participate with the community around Exhibit.

Several developers have contributed additional functionality to the Exhibit code base. The “Exhibit SVN History” *exhibit* shows, that about 16 developers contributed additional code or bug-fixes.

Exhibit's community very much relies on David Huynh's presence. This works well, because at the moment he is available. But as David Huynh stated on the newsgroup, he might not be able to dedicate as much time to Exhibit development as today in the future (HUYNH, 2008a). It is very critical for Exhibit to keep its community lively. I think, the most important thing is, that the code base is documented well, so that more developers can be motivated to maintain and extend it. This leads to the next discussion point, documentation:

7.2 Documentation

Dealing with the topic “Technical Documentation” showed, that there exists a considerable amount of documentation, available both for users and developers, interested in Exhibit. On the other hand, some features are not or just rudimentary documented and the Exhibit wiki also contains several outdated documentation entries. This of course is due to the fact, that Exhibit is quickly developing.

David Huynh's recent comment to the SIMILE Widgets Google Group states the current state in a very good way:

“It's unfortunate that I'm the only full-time developer on Exhibit, Babel, Timeline, Seek, Potluck, Backstage. Sometimes I also provide technical support for Solvent, Piggy Bank, Longwell, and chimed in on Crowbar. I also have other duties and interests. And documentation isn't a very appealing task compared to research and coding ... The hope is that our enthusiastic users might spare a minute to help editing our wiki and add tips for other users.

And this hope has worked out in a number of cases. Tom Woodward and Brian Croxall have

made excellent tutorials on Exhibit and Timeline; we're very grateful! And since our code move to Google Code we have gotten a number of code contributions, too. That's very exciting.

Another hope is to find funding for a real full-time developer, one who doesn't get distracted with wild research tangents like I do. :-) “

(HUYNH, 2008b)

This of course underlines that Exhibit is facing a challenge to find contributors supporting the documentation process. But it also states, that several community members have already provided various contributions.

A consistent and up-to-date API documentation for Exhibit would help a lot. A detailed analysis of the issues, preventing the generation of proper API documentation for Exhibit using JsDoc toolkit was out of this thesis' scope. But it can be assumed, that structural changes to Exhibit's source code could enable JsDoc toolkit to provide an API documentation. I think, the current code base is well structured, so the required, structural changes for the JsDoc toolkit could be applied straight-forward. On the other hand, I can't estimate the exact work effort this step would require. Also it has to be considered, that the Exhibit code base is continuously evolving, so such changes would need high-quality coordination of the developers. Adding code documentation itself would be a necessary, sequent step in order to add meanings to such an API documentation.

If more developers start to work continuously on Exhibit, the use of an Issue Tracker would be crucial, too. Currently, the official issue tracker, hosted at the SIMILE website, contains 44 open issues, but they are not processed: http://simile.mit.edu/issues/browse/EXHIBIT. To integrate other developers more easily, I propose to move over to the SIMILE Widgets Google Code Issue tracker: http://code.google.com/p/simile-widgets/issues/list

7.3 Performance

David Huynh (2007a, p.70ff) evaluated Exhibit's performance as a part of his thesis. The results indicate, that Exhibit's client-side approach works in a common browser environment. Due to the fact, that all data has to be transferred to the client and processed by it, Exhibit is limited in terms of scalability.

In the current test-setup of OpenEvents, the platform displays up to 350 events. If OpenEvents generates an *exhibit,* containing all the 350 events, the *facet* filters become quite slow (2-3 seconds to filter on an *expression*, 4-5 to release the filter again). Also initial load time high, because the browser has to fetch a lot of JavaScript data, especially if Exhibit API extensions are used. It has to be states, that Exhibit already optimizes its JavaScripts by bundling several source file into one and renaming local variables to shorter names.

To overcome performance issues for huge data sets, David Huynh already started another project, Backstage. Backstage is the codename for a prototype that extends Exhibit in “out-sourcing” data to a server-side component (HUYNH, 2007e). The most recent example, available is a “*quick attempt at crawling the (publicly accessible) exhibits and showing their data in a Backstage installation*” (HUYNH, 2008c):
http://people.csail.mit.edu/dfhuynh/misc/backstage-demo-the-ex.html

8 Listings

8.1 List of Illustrations

8.2 List of References

Papers:

- HUYNH D. F., 2007. User Interfaces Supporting Casual Data-Centric Interactions on the Web [Online]. Massachusetts Institute of Technology.
  Available from: http://davidhuynh.net/media/thesis/thesis.php [cited 14 May 2008]
- HUYNH D. F. KARGER D. R. MILLER R. C., 2007. Exhibit: Lightweight Structured Data Publishing [Online]. MIT Computer Science and Artificial Intelligence Laboratory.
  Available from: http://davidhuynh.net/media/papers/2007/www2007-exhibit.pdf [cited 14 May 2008]

Web Sites:

- World Wide Web Consortium, 2008, World Wide Web Consortium – Web Standards [Online]. World Wide Web Consortium.
  Available from: http://www.w3.org/ [Accessed 31 May 2008].
- SIMILE Project, 2008, SIMILE Project [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/ [Accessed 31 May 2008].
- STEFANO, 27 February 2007, SIMILE:About [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/SIMILE:About [Accessed 31 May 2008].
- World Wide Web Consortium, 2008, W3C Semantic Web Activity [Online]. World Wide Web Consortium.
  Available from: http://www.w3.org/2001/sw/ [Accessed 31 May 2008].
- HUYNH, D., 20 September 2006, Exhibit [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/mediawiki/index.php?title=Exhibit&oldid=2533 [Accessed 31 May 2008].
- Massachusetts Institute of Technology. 2008, All Articles with prefix Exhibit [Online].
  Available from: http://simile.mit.edu/mediawiki/index.php?title=Special%3APrefixindex&from=exhibit&namespace=0 [Accessed 31 May 2008]
- Massachusetts Institute of Technology. 2008, Exhibit/2.0 – History [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/mediawiki/index.php?title=Exhibit/2.0&action=history [Accessed 31 May 2008]
- Massachusetts Institute of Technology. 2008, Exhibit/2.0 [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/exhibit [Accessed 31 May 2008]
- DUOSHUTE, 2 May 2008, Exhibit/How to make an exhibit from data fed directly from a Google Spreadsheet [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/How_to_make_an_exhibit_from_data_fed_directly_from_a_Google_Spreadsheet [Accessed 31 May 2008]

- DALE, R., 26 March 2008, Exhibit/Lenses, Views and Facets [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/Lenses_Views_and_Facets [Accessed 31 May 2008].
- DALE, R., 26 March 2008, Exhibit/For Authors [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/For_Authors [Accessed 31 May 2008].
- CAMFIELD, J., 6 February 2008, Exhibit/2.0/Timeline View [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/2.0/Timeline_View [Accessed 31 May 2008].
- PFRED60, 13 March 2008, Exhibit/2.0/Tabular View [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/2.0/Tabular_View [Accessed 31 May 2008].
- LDD, 8 July 2007, Exhibit/FAQ [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/FAQ [Accessed 31 May 2008].
- KARGER, 6 July 2007, Exhibit/Dynamic URLs [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/Dynamic_URLs [Accessed 31 May 2008].
- ANDRISI, 4 February 2008, Exhibit/2.0/Coders [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/2.0/Coders [Accessed 31 May 2008].
- HUYNH, 17 June 2007, Exhibit/2.0/Coordinator [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/2.0/Coordinators [Accessed 31 May 2008].
- SABERHAGEN427, 23 January 2008, Exhibit/2.0/Formats [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/2.0/Formats [Accessed 31 May 2008].
- DASJO, 22 May 2008, Exhibit/Creating, Importing and Managing Data [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/Creating%2C_Importing%2C_and_Managing_Data [Accessed 31 May 2008].
- Prototype Core Team, 2008, Prototype JavaScript framework: API.Prototype API Documentation [Online]. Prototype Core Team.
  Available from: http://www.prototypejs.org/api [Accessed 31 May 2008].
- The Dojo Foundation, 2008, dojo | Dojo Toolkit [Online]. The Dojo Foundation.
  Available from: http://api.dojotoolkit.org/ [Accessed 31 May 2008].
- MICMATH, 2008, jsdoc-toolkit [Online]. Google Code.
  Available from: http://code.google.com/p/jsdoc-toolkit/ [Accessed 31 May 2008].
- Sun Microsystems, Inc., http://java.sun.com/j2se/javadoc/ 2008, Javadoc Tool Home Page [Online]. Sun Microsystems, Inc..
  Available from: [Accessed 31 May 2008].
- Wikipedia, 31 March 2008, Austrian Service Abroad - Wikipedia, the free encyclopedia [Online]. Wikimedia Foundation, Inc.
  Available from: http://en.wikipedia.org/wiki/Auslandsdienst [Accessed 31 May 2008].
- Österreichischer Auslandsdienst, 2008, Österreichischer Auslandsdienst – WebNeu [Online]. Österreichischer Auslandsdienst.
  Available from: http://webneu.auslandsdienst.at/ [Accessed 31 May 2008].
- Plone Foundation, 2008, Plone CMS: Open Source Content Management [Online]. Plone Foundation.
  Available from: http://plone.org/ [Accessed 31 May 2008].
- Zope Coorporation, 2008, Zope.org [Online]. Zope Coorporation.
  Available from: http://www.zope.org/ [Accessed 31 May 2008].
- ASPELI, M., 14 January 2008, Zope 3 browser views [Online]. Plone Foundation.
  Available from: http://plone.org/documentation/tutorial/customization-for-developers/zope-3-browser-views [Accessed 31 May 2008].
- RICHTER, S., 17 November 2004, Zope 3 wiki ZCML [Online]. Zope Coorporation.
  Available from: http://wiki.zope.org/zope3/ZCML [Accessed 31 May 2008].
- IPPOLITO, B., 2008, simplejson [Online]. Google Code.
  Available from: http://code.google.com/p/simplejson/ [Accessed 31 May 2008].
- IPPOLITO, B., 4 May 2008, simplejson 1.9 [Online]. Google Code.
  Available from: http://simplejson.googlecode.com/svn/tags/simplejson-1.9.1/docs/index.html [Accessed 31 May 2008].

- HUYNH, D., 11 April 2007, Exhibit/Template/JSON Data File [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/Template/JSON_Data_File [Accessed 31 May 2008].
- ROSSMAJER, B., 28 October 2007, Exhibit with Included Data [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/mail/ReadMsg?listId=10&msgId=22059 [Accessed 31 May 2008].
- SIMON, 9 June 2008, METAL [Online]. Zope Coorporation.
  Available from: http://wiki.zope.org/ZPT/METAL [Accessed 31 May 2008].
- DE VITIS, M., 16 March 2008, Macros and Slots [Online]. Plone Foundation.
  Available from: http://plone.org/documentation/tutorial/zpt/macros-and-slots [Accessed 31 May 2008].
- EVAN, 8 March 2008, TALES [Online]. Zope Coorporation.
  Available from: http://wiki.zope.org/ZPT/TALES [Accessed 31 May 2008].
- Smart Information Systems, 2008, OpenEvents [Online]. Smart Information Systems.
  Available from: http://openevents.at [Accessed 31 May 2008].
- HOLLERER, S., KLAUSZ, M. DABERNIG, J., 2008, OpenEvents – Technische Spezifikation (Programmlogik und User Interface) [Online]. Smart Information Systems.
  Available from: http://openevents.at/docs/OpenEvents_Spezifikation.pdf [Accessed 31 May 2008].
- PEREZ, P. L., 2008, jsonmarshaller [Online]. Google Code.
  Available from: http://code.google.com/p/jsonmarshaller [Accessed 31 May 2008].
- PEREZ, P. L., 18 March 2008, Introduction - jsonmarshaller [Online]. Google Code.
  Available from: http://code.google.com/p/jsonmarshaller/wiki/Introduction [A ccessed 31 May 2008].
- HUYNH, D., 17 June 2007, Exhibit/2.0/Coordinators [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/wiki/Exhibit/2.0/Coordinators [A ccessed 31 May 2008].
- HUYNH, D. 18 February 2008, My departure + the future of Timeline and Exhibit [Online]. Massachusetts Institute of Technology.
  Available from: http://simile.mit.edu/mail/ReadMsg?listName=General&msgId=23992 [A ccessed 31 May 2008].
- HUYNH, D. 28 May 2008, Re: Pivot View Totals? [Online]. Google Groups.
  Available from: http://groups.google.com/group/simile-widgets/msg/14131135d5419d9e [A ccessed 31 May 2008].
- HUYNH, D. 7 Feb 2007, scaling up Exhibit – an early experiment. [Online]. Massachusetts Institute of Technology.
  Available from: http://www.nabble.com/scaling-up-Exhibit---an-early-experiment-p15339573.html [Accessed 31 May 2008]
- HUYNH, D. 25 May 2008, The Exhibition – merging all exhibits. [Online]. Massachusetts Institute of Technology.
  Available from: http://groups.google.com/group/simile-widgets/msg/b379cdeed32269f9 [A ccessed 31 May 2008].