Bibsonomy Webservice API: Developer's Guide

Manuel Bork manuel.bork@uni-kassel.de

September 14, 2007

Abstract

Bibsonomy[1] is a social bookmark and publication sharing system developed on the chair of knowledge and data engineering[2] of the university of Kassel [**?**] [**?**]. The available document is a guidance how to develop applications using the Bibsonomy webservice API.

1 Introduction

Bibsonomy provides a webservice using *Representational State Transfer* (REST), a software architectural style for distributed hypermedia systems. The term originated in a 2000 doctoral dissertation about the web written by Roy Fielding [**?**], one of the principal authors of the HTTP protocol specification. This article is intended for developers who want to develop applications which interact with Bibsonomy.

2 The Java client library

Though it's possible to directly interact with Bibsonomy's webservice, as described in section 3, the recommended way is to use the provided client library, written in Java. The Java library encapsulates each accessing method provided by the webservice (see appendix A) with a corresponding class.

2.1 Getting started

The client library is free for download via HTTP from http://www.bibsonomy.org/.... The client library consists of multiple jars which must be added to your Java project's classpath. Those jars are:

Bibsonomy webservice The webservice client itself consists of three jars: bibsonomy-client.jar (client library), bibsonomy-data-model.jar (bibsonomy data model), bibsonomy-shared.jar (shared library).

Apache HTTP-Client The HTTP client libraries by the apache software foundation are used for the HTTP communication stuff[3]: commons-httpclient.jar.

Java Architecture for XML Binding (JAXB) The client uses JAXB to (de)serialize the XML exchange documents[4]: jaxb-api.jar.jar, jaxb-impl.jar, jsr173.jar, activation.jar

Fujaba associations This library is used by the data model for encapsulating the n:n and n:m associations[5]: assocs.jar.

[1] http://www.bibsonomy.org
[2] http://www.kde.cs.uni-kassel.de
[3] http://jakarta.apache.org/commons/httpclient/
[4] http://java.sun.com/webservices/jaxb/
[5] http://www.fujaba.de

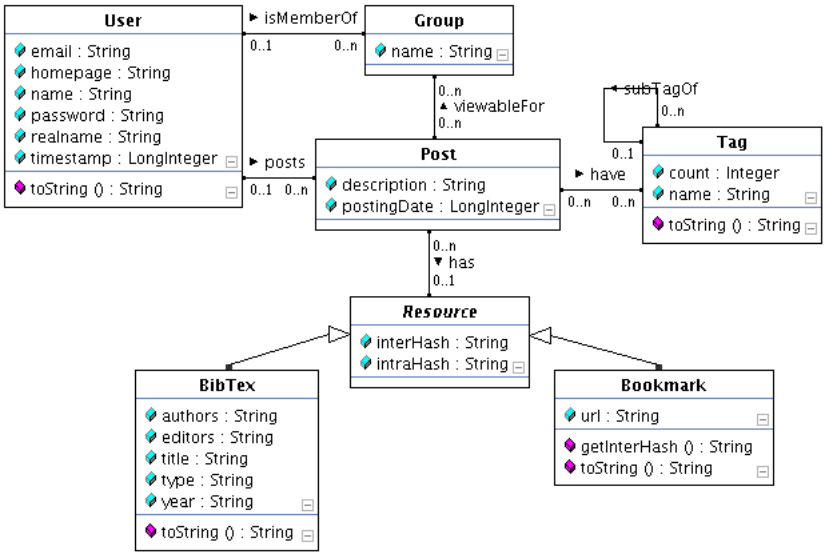

Figure 1: UML class diagram of the Bibsonomy data model

2.2 Using the client library

Using the client library is rather simple. At first one has to instantiate an object of type *Bibsonomy* and set the user's credentials on it. This object can be reused during program run. As already discussed, there exist classes encapsulating each provided webservice method. These classes are derived from *AbstractQuery* and request in their constructor parameters (usually model data objects, see the package *org.bibsonomy.model*, and see figure 1 for the uml class diagram of the Bibsonomy data model) - please refer to the Java doc of the corresponding constructor. After instantiating an appropriate *AbstractQuery* just execute it by invoking the *executeQuery* method of the Bibsonomy object with the query object as parameter. If your query object fetches a list of objects, for example a list of posts, you can also use the overloaded *executeQuery* method and register a callback object (use the *ProgressCallback* interface). The return values of your query are available via the *getResult* and *getHttpStatusCode* methods of the *AbstractQuery*.

Here is a simple example of how to use the client library:

```
// create webservice client object
Bibsonomy bib = new Bibsonomy( "myUserName", "mySecretPassword" );

// instantiate query object
GetPostsQuery gpq = new GetPostsQuery();
// some queries can be parametrized
// in this example we want to fetch only bibtex entries
gpq.setResourceType( ResourceType.BIBTEX );

try
{
   // perform query
   bib.executeQuery( gpq );

   // on success, read results
   if( gpq.getHttpStatusCode() == 200 )
   {
      List<Post> posts = gpq.getResult();
```

```
      for( Post post: posts )
      {
         /*
          * now do something with your posts :)
          */
      }
   }
}
catch( ErrorPerformingRequestException e )
{
   /*
    * happens on network failure for example
    */
}
```

A whole example application for managing bookmarks and publications, demonstrating the webservice client library, is available for download at http://www.bibsonomy.org/....

3 Direct use of the REST webservice

If you intend to use the Bibsonomy webservice with another programming language than Java, you are welcome to develop your own client library communicating directly with the REST webservice. In the REST architecture's philosophy each accessible object gets its own URI. The accessible objects of Bibsonomy are *users*, *groups*, *posts* and *tags*. See [?] for an introduction into their semantics. The exposed URIs are summarized in appendix A. REST webservices exchange structured data; so all exchange documents must be valid to a given XML schema. Appendix B shows an extract of this schema, it can be viewed in whole on https://www.bibsonomy.org/BibWiki/RestApiUrls **TODO: move documentation to a public website!**. If you succeed in writing your own client library, please let us know and send us your files, so that other programmers can benefit from your work. Thank you in advance.

A URL pattern

For the uniform addressing and identification a URL pattern was specified, which one can introduce oneself like a file system. The individual inquiries can be partly supplemented with attributes of the form *key=value*.

Description	HTTP-Method	URL
List of all existing users	GET	/users
Create user	POST	/users
An user's details	GET	/users/[username]
Change an user's details	PUT	/users/[username]
Delete an user	DELETE	/users/[username]
List of a user's posts	GET	/users/[username]/posts
		?tags=[t1+t2+...+tn]
		?resourcetype=(bibtex|bookmark)
Add a new post	POST	/users/[username]/posts
A post's details	GET	/users/[username]/posts/[resourceHash]
Change of a post	PUT	/users/[username]/posts/[resourceHash]
Delete a post	DELETE	/users/[username]/posts/[resourceHash]
List of all groups	GET	/groups
Add a new group	POST	/groups

Description	HTTP-Method	URL
A group's details	GET	/groups/[groupname]
Change a group's details	PUT	/groups/[groupname]
Delete group	DELETE	/groups/[groupname]
List of a group's members	GET	/groups/[groupname]/users
Add a group member	POST	/groups/[groupname]/users
Remove a group member	DELETE	/groups/[groupname]/users/[username]
List of all tags	GET	/tags ?filter=[regex] ?(user|group|viewable)=[username/groupname]
A tag's details	GET	/tags/[tag]
Substitute all occurrences of t1,t2,.. by T1,T2,..	PUT	/substitutetags?from=[t1+t2+..]&to=[T1+T2+..]
List of all posts	GET	/posts ?tags=[t1+t2+...+tn] ?resourcetype=(bibtex|bookmark) ?(user|group|viewable)=[username/groupname] ?resource=[hash]
List of all new posts	GET	/posts/added ?resourcetype=(bibtex|bookmark)
List of all popular posts	GET	/posts/popular

All URLs can be supplemented by these attributes (use *start* and *end* only with lists):

```
?format=(xml|rdf|html)
?start=[int], starting at 0, default 0
?end=[int], starting at 0, default 19
```

Tags can be customized the following way:

```
->[tag]    tag and its direct children
-->[tag]   tag and its children (transitive)
[tag]->    tag and its direct parents
[tag]-->   tag and its parents (transitive)
<->[tag]   tag and its correlated tags
```

B XML schema

For data exchange XML documents are used. The individual exchange documents are valid to a XML schema, which is represented for the *posts* in part here. The complete schema with examples and explanations can be viewed at https://www.bibsonomy.org/BibWiki/RestApiUrls **TODO: move documentation to a public website!**.

```
<!-- a post -->
<xsd:complexType name="PostType">
  <xsd:sequence>
    <xsd:element name="user" type="UserType"/>
    <xsd:element name="group" type="GroupType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="tag" type="TagType" maxOccurs="unbounded"/>
    <xsd:choice>
      <xsd:element name="bookmark" type="BookmarkType"/>
      <xsd:element name="bibtex" type="BibtexType"/>
    </xsd:choice>
   </xsd:sequence>
   <xsd:attribute name="description" type="xsd:string"/>
</xsd:complexType>
<!-- a bookmark -->
<xsd:complexType name="BookmarkType">
  <xsd:attribute name="url" type="xsd:anyURI" use="required"/>
</xsd:complexType>
<!-- a bibtex -->
<xsd:complexType name="BibtexType">
  <xsd:attribute name="title" type="xsd:string" use="required"/>
```

```
  <xsd:attribute name="authors" type="xsd:string"/>
  <xsd:attribute name="editors" type="xsd:string"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="year" type="xsd:string"/>
</xsd:complexType>
<!-- a user -->
<xsd:complexType name="UserType">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<!-- a tag -->
<xsd:complexType name="TagType">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
```

References