GdataServer HowTo

Configure GData - Server (gdata-config.xml)

service section

The service section defines a specific feed implementation with a defined structure. The server can provide arbitrary kinds of feeds (and entries). A Feed is represented internally as a sub class of `com.google.gdata.data.BaseFeed` and can contain entries defined by subclassing `com.google.gdata.data.BaseEntry`. To provide a certain feed service the type of feed and entry must be definded in the configuration file below. Once a service has been created (after first startup) the feed and entry class can not be changed in sense of a "down cast". Users can only insert / update entries of the structure the entry class represents.

The service name has a special meaning within the service. A service is accessed via `http://yourDomain.com:port/gdata-server/servicename/feedid` so the service name identifies the name of the service endpoint.

The class name specified in the *extension-profile* element is used to generate and read feeds and entries. Extensions of feed or entry types will be declared to an instance of the extension profile class while runtime.

Index

The GData - Server creates a single index for each service which allows moving a specific service to another server. This section has to be configured for each service.

Index attributes

- *useTimedIndexer* - `true|false` if set to `true` the indexer will idle for the defined idle time and run a commit if no modification to the index occured. If set to `false` the indexer will commit after the amount of documents configured in the `commitAfterDocuments` attribute.
- *indexerIdleTime* - the time in seconds the indexer waits until run a automatic commit. if no document has to be written to the index the called commit will not force a release of a new index searcher.
- *optimizeAfterCommit* - the amount of commit without performing an index optimization. Auto commits by a timed indexer without any documents to write will be omitted.
- *commitAfterDocuments* - the amount of document after the indexer runs a commit and releases a new index searcher including timed indexer.

For information about the effect of these attribute see IndexWriter JavaDoc and contact the lucene documentation.

```
<useCompoundFile>true</useCompoundFile>
<mergeFactor>10</mergeFactor>
<maxBufferedDocs>1000</maxBufferedDocs>
<maxMergeDocs>2147483647</maxMergeDocs>
<maxFieldLength>10000</maxFieldLength>
```

configure index fields

To keep indexing of entries as flexible as possible each element of a feed entry which should be searchable has to be configured in the index schema. The name of a field matches the get parameter in the http search query to search a feed. The element or rather the node to use for indexing will be defined by an xpath expression which can also point to an attribute. The indexer will use the text content of the Node to create the index field. **XPath expressions are always used in a ATOM 1.0 context.**

```
<field name="title" boost="2.0" type="text">
  <path>/entry/title</path>
  <analyzer>
   org.apache.lucene.analysis.standard.StandardAnalyzer
  </analyzer>
  <store>YES</store>
  <index>TOKENIZED</index>
</field>
```

To define a specific analyzer for a field use the analyzer element, this analyzer will be used for indexing and searching. If the class can not be found or is not a type of `org.apache.lucene.analysis.Analyzer` the server will fail at startup. The elements `store` and `index` take values of constants defined in Store and Index. These elements are not requiered the default value is `Index.TOKENIZED` and `Store.NO`. Values must be upper case.

The attribute `type` describes the format or rather the type of the content to index. As indexing dates needs another strategy than indexing plain text content.

Predefined types:

- text - Plain text content
- html - removes all html tags from the content string
- xhmtl - same behaviour as html
- mixed - detects the content type (html, xhtml or text)
- gdatadate - RFC 822 date format format
- keyword - treats the content as a keyword (Index.UN_TOKENIZED)
- category - entry category

If no predefined strategy matches, custom implementation can be created by subclassing ContentStrategy. Fields using custom content strategy implementation need to be configured using the `custom` element:

```
<custom name="custom" boost="1.0" >
  <path>/entry/customelement</path>
  <field-class>
    org.apache.lucene.gdata.search.analysis.ExtendContentStrategy
  </field-class>
  <analyzer>
    org.apache.lucene.analysis.SimpleAnalyzer
  </analyzer>
</custom>
```

Mandatory Fields

The GData protocol defines a small query language with 4 defined get parameter. (see Query definition) To keep the server compatible with the client API and the protocol each index schema must contain these field names.

- updated - the updated element of a atom feed
- category - the category of an entry
- author - the entry author
- q - main query (uses the field set in the `defaultSearchField` attribute

These fields are predefined in the `gdata-config.xml` file and should not be changed to keep your server compatible with the GData protocol.

```
<?xml version="1.0" encoding="UTF-8"?>
<gdata>
<!--
     the poolsize is the size of the extension profile pool for a specific service. Pooling the
     extension profile instances improves performance and enables reusing the declared
     extensions. Increase that if you have to handle lots of concurrent requests.
-->

  <service name="feed" poolSize="20" public="true">
    <feed-class>com.google.gdata.data.Feed</feed-class>
    <entry-class>com.google.gdata.data.Entry</entry-class>
    <extension-profile>
      com.google.gdata.data.ExtensionProfile
    </extension-profile>
    <index-schema defaultSearchField="content">
      <index useTimedIndexer="true" indexerIdleTime="120" optimizeAfterCommit="5" commitAfterDocuments="10">

        <!--
        The default analyzer will be used if no analyzer is specified for a configured index field
        -->
        <defaultAnalyzer>
          org.apache.lucene.analysis.standard.StandardAnalyzer
        </defaultAnalyzer>

        <!--
        The index location can be the same for all configured services.
        The index controller will create a subfolder for  each service.
        -->
        <indexLocation>/tmp</indexLocation>
        <useCompoundFile>true</useCompoundFile>
        <mergeFactor>10</mergeFactor>
        <maxBufferedDocs>1000</maxBufferedDocs>
        <maxMergeDocs>2147483647</maxMergeDocs>
        <maxFieldLength>10000</maxFieldLength>
      </index>
      <field name="title" boost="2.0" type="text">
        <path>/entry/title</path>
        <analyzer>
            org.apache.lucene.analysis.standard.StandardAnalyzer
        </analyzer>
        <store>YES</store>
        <index>TOKENIZED</index>
      </field>
      <mixed name="content" boost="1.0">
        <path>/entry/content</path>

        <!--
        This xpath must point to an attribute defining the content type.
        Either html, xhtml or text. If the content type can not be detected
        text will be used as default
        -->
        <contenttype>/entry/content/@type</contenttype>
        <analyzer>
          org.apache.lucene.analysis.StopAnalyzer
        </analyzer>
        <store>YES</store>
        <index>TOKENIZED</index>
      </mixed>
  </service>
```

Server Components

Functional parts of the GData - Server are devided in plugable components to enable users to implement their own storage, indexer or any other available component.

Available components:

- Storage - implements org.apache.lucene.gdata.storage.StorageController
- Search - implements org.apache.lucene.gdata.search.SearchComponent
- RequestHandler- implements org.apache.lucene.gdata.servlet.handler.RequestHandlerFactory
- Service - implements org.apache.lucene.gdata.server.ServiceFactory
- Authenticaton - implements org.apache.lucene.gdata.server.authentication.AuthenticationController

These components or rather the implementation will be loaded at server startup and registered in the core component of the server as a singleton. To access components at runtime the core component provides a JNDI like lookup service. The Registry calles the components initialize method while registering and the destory method on shutdown to release all resources.

To register a configured class as a component the class must be annotated with the public `org.apache.lucene.gdata.server.registry.Component` which is visible at runtime.

```
  <server-components>
     <component>
      <class>
        org.apache.lucene.gdata.storage.db4o.DB4oController
      </class>
      <configuration>
        <property name="runAsServer">true</property>
        <property name="port">0</property>
        <property name="host">localhost</property>
        <property name="filePath">
        /tmp/
        </property>
        <property name="containerPoolSize">10</property>
        <property name="user">entw_apache</property>
        <property name="password">Eh2aSZ</property>
        <property name="useWeakReferences">true</property>
      </configuration>
      </component>
      <component>
        <class>
          org.apache.lucene.gdata.servlet.handler.DefaultRequestHandlerFactory
        </class>
      </component>
      <component>
        <class>
        org.apache.lucene.gdata.search.index.IndexController
        </class>
      </component>
      <component>
        <class>org.apache.lucene.gdata.server.ServiceFactory</class>
      </component>
      <component>
       <class>
          org.apache.lucene.gdata.server.authentication.BlowfishAuthenticationController
       </class>
      <configuration>
       <property name="key">cryptKey</property>
       <property name="loginTimeout">60</property>
      </configuration>
    </component>
  </server-components>
</gdata>
```

Create a custom component

Providing a custom component itself looks quiet like an easy job at first glance. The following example shows how an custom implementation of the request component. To provide the actual component you just have to extend the abstract base class `RequestHandlerFactory` or extend the default implementation and override the method you want to customize.

```
import org.apache.lucene.gdata.server.registry.Component;
import org.apache.lucene.gdata.server.registry.ComponentType;
import org.apache.lucene.gdata.servlet.handler.GDataRequestHandler;
import org.apache.lucene.gdata.servlet.handler.RequestHandlerFactory;

/*
 *The Component annotation defines the ComponentType for this class will
 *will be registerd in the registry at server startup.
 */
@Component(componentType = ComponentType.REQUESTHANDLERFACTORY)
public class CustomRequestHandlerFactory extends RequestHandlerFactory {

    public CustomRequestHandlerFactory() {
        super();
   }

    @Override
    public GDataRequestHandler getEntryUpdateHandler() {

        return null;
    }
...
```

The actual job you have to do is to implement the methodes you want to customize which can turn out in a fair bit of work. To use your custom implementation the class has to be configured as a component in the `gdata-config.xml` file

```
 <component>
    <class>
      org.apache.lucene.gdata.servlet.handler.DefaultRequestHandlerFactory
    </class>
 </component>
```

If you want to configure some parameters for your custom implementation the registry provides a kind of a property setter service which allows you to define your properties inside the component definition. This keeps all configuration in one single configuration file. Properties are defined by the name of the setter method so to set a value with the name "title" your component class needs a setter method with the following signature "public void setTitle (String title)". The type of the property must either be a string, a primitive data type or must provide a string constructor.

```
 <component>
    <class>
      org.apache.lucene.gdata.servlet.handler.DefaultRequestHandlerFactory
    </class>
    <configuration>
      <property name="title">yourName</property>
    </configuration>
 </component>
```

If a property is mandatory and must be set at startup or initialisation of a component you can annotate the setter method using the `org.apache.lucene.gdata.server.registry.configuration.Requiered` annotation. This causes the server to fail on startup if this property is not configured.

```
...
   @Requiered
   public void setTitle(String title){
      this.title = title;
   }
```

Server Administartion

Currently there is no web based adiminstration interface provided by the GData - Server but that does not mean the server can not be administrated. All administration goes via the http protocol using a basic REST approach using POST to insert, PUT to update and DELETE to delete. Currently there is not method lists all created feeds or users but that will be added in later development.

The following examples using the Jakarta HTTP-Client lib to send http requests to the server.

Authenticate an existing account

Requiered jars:

- gdata-client.jar

```
 /*
  * The protocol can either be http or https, depends on your server configuration.
  * The name of the application is just a meta data field an can be ommited.
  */
  GoogleService service = new GoogleService("feedId","yourapplicationname", "http", "localdomain:port/gdata-
server");
 /*
  * ServiceType should be configured in the gdata-config.xml
  * Username and password for the admin account are the default values
  * and should be changed
  */
  String authToken = service.getAuthToken("administrator","password", null,null,"servicetype","
yourapplicationname");
```

You can find more details on using the client api here.

Create/update/delete an account

This example shows an update of the adminstrator account (set by default). To delete or create an account just alter the http method to DELETE or POST.

```
  String accountAdminEndpoint = "http://www.yourdomain.com/gdata-server/admin/account";
  /*
   * XML Account format to send in the http request body
   */
  String account = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
  "<account>" +
    "<account-name>administrator</account-name>" +
    "<password>yourpassword</password>" +
    "<account-role>15</account-role>" +
    "<account-owner>" +
      "<name>your name</name>" +
      "<email-address>admin@apache.org</email-address>" +
      "<url>http://www.apache.org</url>" +
    "</account-owner>" +
    "</account>";

  RequestEntity feedReqeustEnt = new StringRequestEntity(account);
  /*
   * Put method to update the account
   */
  PutMethod httpMethod = new PutMethod(accountAdminEndpoint);
  /*
   * you need to authenticate before adding a new feed. Authentication
   * will return an authentication token.
   */
  httpMethod.addRequestHeader(new Header("Authorization",
  "GoogleLogin auth=yourAuthenticationstring"));
  httpMethod.setRequestEntity(feedReqeustEnt);

  httpMethod.setQueryString("account=yourAccountName&service=configuredServiceName");

  HttpClient client = new HttpClient();
  try{
    client.executeMethod(httpMethod);
  }finally {
    httpMethod.releaseConnection();
  }
```

Account Roles

GData - Server uses a simple role system to grant users access to certain services. This roles are represented by an integer or rather the bit pattern of the integer.

- User role - Integer: 1
- Entry Administrator role - Integer: 2
- Feed Administrator role - Integer: 4
- User Administrator role - Integer: 8

To give a certain user more than one role you just have to add the desired role to the base role.

Entry and Feed Administrator is represented by the Integer value of 6. The highest role is 15. For infromation on the different roles see the FAQ

Feed administration

Insert / Update a feed

The following code shows how a simple main method could be used to insert a new feed. To update a feed all you need to do is to change the http method into put.

Requiered jars:

- commons-httpclient.jar
- commons-codec.jar
- gdata-client.jar

```
String feedAdminEndpoint = "http://www.yourdomain.com/gdata-server/admin/feed";
com.google.gdata.data.Feed feed = new Feed();
feed.setId("myFeedId");
feed.setTitle(new PlainTextConstruct("new feed"));
// ... set all properties

// generate the xml to send to the server
StringWriter writer = new StringWriter();
com.google.gdata.util.common.xml.XmlWriter gdataXmlWriter = new XmlWriter(writer);
feed.generateAtom(gdataXmlWriter, new com.google.gdata.data.ExtensionProfile());

String feedXmlString = writer.toString();

RequestEntity feedReqeustEnt = new StringRequestEntity(feedXmlString);
/*
 * Use Put for update and Delete for delete
 */
PostMethod httpMethod = new PostMethod(feedAdminEndpoint);
/*
 * you need to authenticate before adding a new feed. Authentication
 * will return an authentication token.
 */
httpMethod.addRequestHeader(new Header("Authorization",
        "GoogleLogin auth=yourAuthenticationstring"));
httpMethod.setRequestEntity(feedReqeustEnt);

httpMethod.setQueryString("account=yourAccountName&service=configuredServiceName");

HttpClient client = new HttpClient();
try{
client.executeMethod(httpMethod);
}finally {
    httpMethod.releaseConnection();
}
```

Delete a feed

Requiered jars:

- commons-httpclient.jar
- commons-codec.jar

```
String feedAdminEndpoint = "http://www.yourdomain.com/gdata-server/admin/feed";
DeleteMethod httpMethod = new DeleteMethod(feedAdminEndpoint);
/*
 * you need to authenticate before adding a new feed. Authentication
 * will return an authentication token.
 */
httpMethod.addRequestHeader(new Header("Authorization",
"GoogleLogin auth=yourAuthenticationstring"));
httpMethod.setRequestEntity(feedReqeustEnt);
httpMethod.setQueryString("feedid=feedIdToDelete");

HttpClient client = new HttpClient();
try{
  client.executeMethod(httpMethod);
}finally {
  httpMethod.releaseConnection();
}
```

Use the client API

This How To / tutorial does not aim to show how use the client API. There is a fairly good tutorial about the GData client API at code.google.com

---