Google App Engine (GAE)

Notes

Author	Faram Khambatta
Created	2012-12-26
Last updated	2013-04-02

Spring 3.x & Velocity on GAE

See Apache Velocity Notes for integration and configuration of Spring and Velocity.

Required Jars (when using JDO 3.0)

compile target in *build.xml* will automatically copy jars from *appengine-java-sdk-xxx/lib/user/* to project's *war/WEB-INF/lib/* . These jars are -

1. All the appengine-xxx.jar jars (3 in number).
2. asm-4.0.jar
3. All the datanucleus-xxx.jar jars (4 in number).
4. geronimo-jpa_2.0_spec-1.0.jar
5. jdo-api-3.0.1.jar
6. jsr107cache-1.1.jar
7. jta-1.1.jar

Manually add the following jars to *war/WEB-INF/lib/* for Spring 3.x and Velocity -

1. commons-collections-3.2.1.jar
2. commons-lang-2.4.jar
3. commons-logging-1.1.1.jar
4. spring-beans
5. spring-context
6. spring-context-support
7. spring-core
8. spring-expression
9. spring-web
10. spring-webmvc
11. velocity-1.7.jar
12. velocity-tools-2.0.jar

All the above jars are required even for a simple Hello World type MVC app. If any of the above jars are missing then project might still compile ok but will fail with missing class exceptions during runtime.

Additionally if *log4j* logging is required then add *log4j* jar.

That makes 25 jars in all for a simple Hello World type MVC app.

If additional functionality is required then more jars might be required.

Access Restrictions

To restrict access to certain URL paths only to users logged in to Google accounts,
in *web.xml*

```
<security-constraint>
   <web-resource-collection>
      <url-pattern>/profile/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>*</role-name>
   </auth-constraint>
</security-constraint>
```

To further restrict access only to *admin* users, in above example, change <*role-name*> from '*' to 'admin'. *Admin* users are registered developers of the application.

For above URLs, GAE will automatically display Google login page and on successful login, will redirect back to app's URL.

Using HTTPS

To make GAE use https for certain URLs,
in *web.xml*

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/profile/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

The Jetty dev server on local machine will ignore above https directive but it will work when deployed on GAE.

Directory structure

WEB-INF/

web.xml
appengine-web.xml
logging.properties
dispatcher-servlet.xml

appengine-web.xml

- WEB-INF/ should contain *appengine-web.xml* over and above *web.xml*. This file contains *app id* and version.
- To enable sessions

  <sessions-enables>true</sessions-enabled>

build.xml

- Change property *sdk.dir* to point to GAE SDK.
- To use JDO 3.0 -
  Modify *copyjars* and *datanucleusenhance* targets as shown in GAE docs.
- Clean before Build -
  In *compile* target, add, before anything else,
  <delete dir="war/WEB_INF/classes"/>

GAE app in NetBeans SDK

General

- Create a *Java Free Form* project.
- Project properties -> Java sources -> Add Folder -> src
  This allows source packages/classes to be added in *Projects* tab.
- NetBeans uses *Ant* build system. *Targets* in *build.xml* are executed.

NetBeans project directory structure

Project directory

- src

- war
    - WEB-INF
        - lib (contains 3rd party and GAE libs)
        - classes (nothing should be directly added to this folder as it is deleted on executing *clean* target.)
        - jsp
- build.xml

Auto-complete and Import

Project properties -> Java sources classpath -> Add Jar/Folder

- Add all 3rd party jars (which are in *lib* folder) to above menu option.
- This will allow auto-complete and import (Ctrl + Shift + I) in NetBeans.

Context Menu items

Project properties -> Build & Run -> Add context menu item

- *Ant* targets (in *build.xml*) show up as entries in context menu when project is right-clicked. By default, only *Build* and *Run* are shown. To add more targets, use above menu. Give each item any arbitrary label and choose required *Ant* target from dropdown list.
- For GAE, useful targets are *update* and *datanucleusenhance*.

Logging

- GAE uses *java.util.logging.Logger*.
- Add *logging.properties* to WEB-INF. It contains a single line -
  .level = WARNING
  (or .level = INFO)
- In *appengine-web.xml*, add

```
<system-properties>
  <property name="java.util.logging.config.file" value="WEB-INF/logging.properties"/>
</system-properties>
```

- To use logger,
  e.g.

```
class MyClass {
        private static final Logger log = Logger.getLogger(MyClass.class.getName());
```

```
    ...
    log.info("Test log msg");
    ...
}
```

Logging using log4j

1. Add *log4j* jar to project i.e. put log4j.jar in yourProjectFolder/war/WEB-INF/lib/
2. Put *log4j.properties* file in yourProjectFolder/src/ i.e. top level source folder.
   compile target in *build.xml* will copy it to war/WEB-INF/classes/ along with other source files.
3. If the project is a *NetBeans* project then add path to *log4j.jar* in
   project properties -> Java sources classpath. This will help in code auto-complete, auto-import, etc.
4. There is no need to add a <*system-properties*> element in *appengine-web.xml* as used for java.util.logging.logger. In fact, if such an element is present then comment it out. Also, there is no need for a *logging.properties* file.

Uploading to GAE

1. cd project directory.
2. appcfg.cmd update war
   (appcfg.cmd is in appengine-sdk-xxx/bin/)
3. If project has been compiled with Java 7 then use this commandline instead -
   appcfg.cmd --use_java7 update war
   (Not required for sdk version 1.7.6 and later)

JDO

Configuration & Build

- Create *jdoconfig.xml* in *src/META-INF/*. *Ant compile* target,when executed, will copy it to *WEB-INF/classes/META-INF/*. This file defines a *persistence-manager-factory* bean.
- After building the app, run *datanucleusenhance* Ant target (in *build.xml*) which does bytecode enhancement. This target is automatically called by *runserver* target.

Usage

- Create a singleton *PMF.java* which returns a singleton *PersistentManagerFactory (PMF)*. This can be used by other classes to persist data. See GAE docs on how to create this class. *PMF.java* gets PMF details from *jdoconfig.xml*.
- Any class needing to persist objects should create a *PersistenceManager* instance from *PMF* and use it for CRUD operations.
  e.g.

```
import some.path.PMF

class MyClass {
        PersistenceManager pm = PMF.get().getPersistenceManager();
    pm.someMethod();
    ...
}
```

- Also see JDO Notes.

Data Types

String	upto 500 chars
gae.Text	upto 1 MB, not indexed
gae.Blob	upto 1 MB byte array, not indexed