

[GWT](#) is probably the most productive technology I have used to build amazing User Interfaces. Now coupled with [Gaelyk](#) you get an amazing combination to get stuff done quicker and productively.

Before I proceed further, if you are new to Gaelyk you might want to read [Java development 2.0: Gaelyk for Google App Engine](#) and the [Gaelyk Tutorial](#). If you are new to GWT you might consider reading the [tutorial](#).

[Download the Gaelyk Template project](#) and extract the files.

Please note this is a very basic tutorial and only scratches the surface of what's possible. It is intended to help you with this [bug](#).

Also this has been written over number of weeks, hence the flow isn't great. When I have time again, I am hoping to fix this and re-release it.

What we want to achieve

In this tutorial I will create a simple GWT Application with Gaelyk Backend. We will use Groovlets instead of the Servlets / RemoteService. Use the Gaelyk wrapper classes for Datastore and Blobstore.

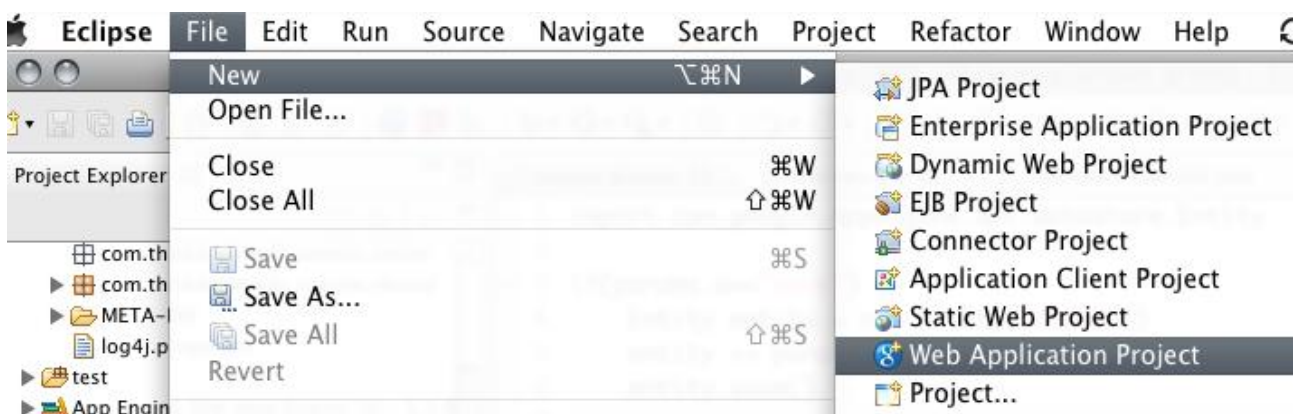
| | |
|--|---|
| Name | <input type="text" value="Ram"/> |
| Email | <input type="text" value="ram@thinkbelievedo.com"/> |
| Photo | <input type="text" value="/Users/ram/Documents/thi"/> <input type="button" value="Browse..."/> <input type="button" value="Upload"/>  |
| <input type="text" value="Ramram@thinkbelievedo.comF0m7AFRdOEKcezpt3rNglg"/> | |
| <input type="button" value="Save"/> | |

Tooling Details

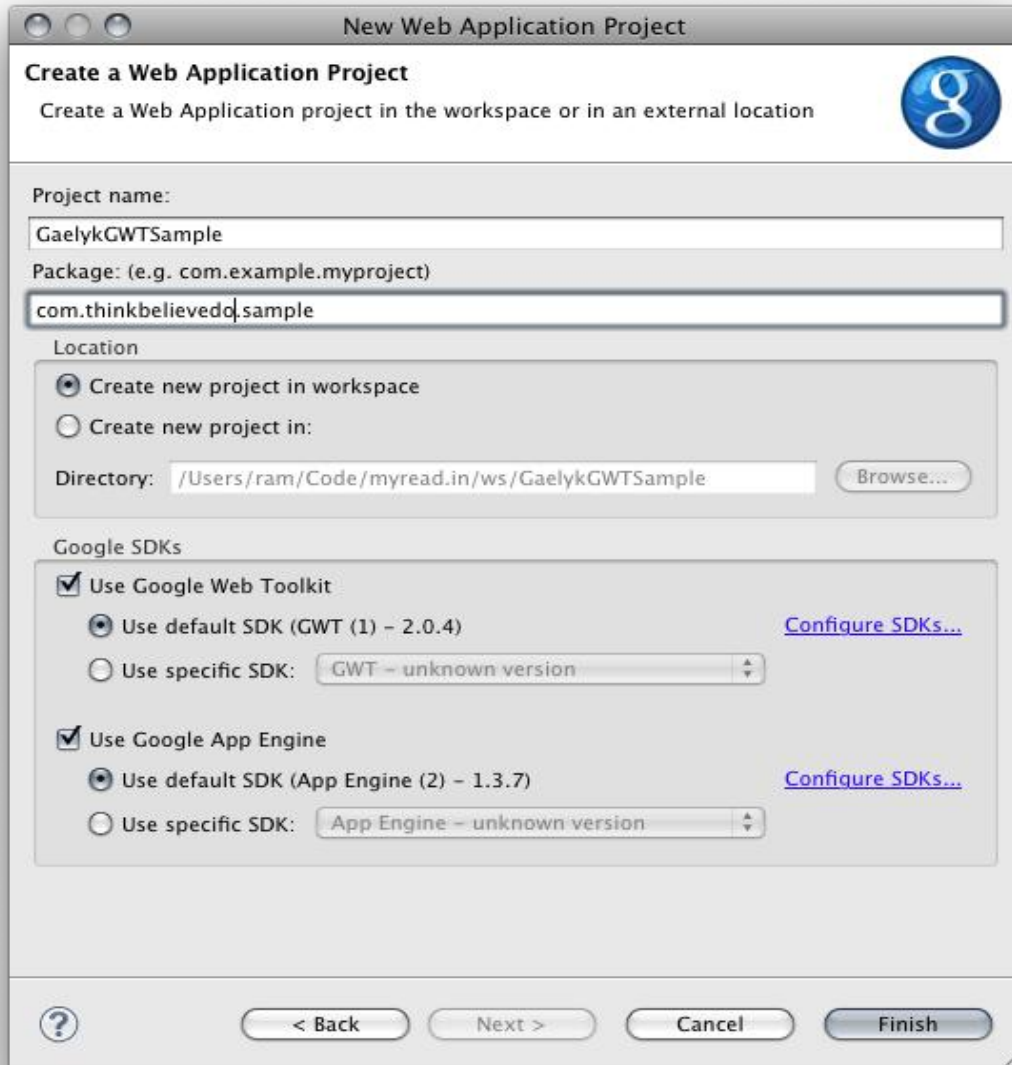
For the sake of this tutorial I have used Eclipse with [Google Plugin for Eclipse](#) and [Groovy Plugin for Eclipse](#) installed.

Starting a new Project

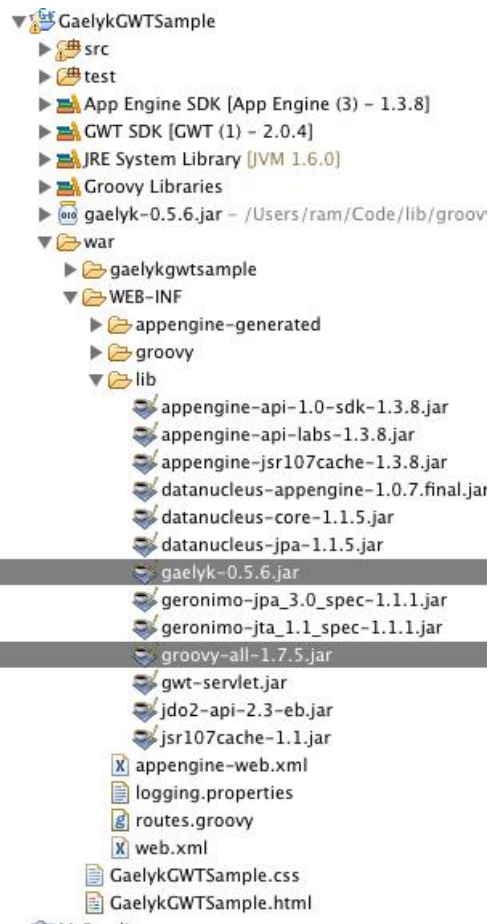
File > New > Web Application Project



Now fill in the details as shown below



This will create a new project. Add the groovy-all-1.7.5.jar & gaelyk-0.5.6.jar from the <templateproject>/lib folder to the WEB-INF/lib as shown below:



Configuration

appengine-web.xml - update the file with the following entries before `</appengine-web-app>` - this excludes all the groovy files to be served as static files.

```
<!-- Exclude Groovlets and templates from the static files -->
<!-- to let App Engine know these files are not just mere resources -->
<static-files>
<exclude path="/WEB-INF/**/*.groovy" />
</static-files>
```

web.xml - remove the default Greetings Servlet (which is provided by the Google Plugin) and replace it with the following

```
<!-- The Gaelyk Groovlet servlet -->
<servlet>
<servlet-name>GroovletServlet</servlet-name>
<servlet-class>groovyx.gaelyk.GaelykServlet</servlet-class>
</servlet>
<!-- Specify a mapping between \*.groovy URLs and Groovlets -->
```

```

<servlet-mapping>
<servlet-name>GroovletServlet</servlet-name>
<url-pattern>*.groovy</url-pattern>
</servlet-mapping>
<filter>
<filter-name>RoutesFilter</filter-name>
<filter-class>groovyx.gaelyk.routes.RoutesFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>RoutesFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>

```

Note: Unlike the a normal Gaelyk App which uses the [Gaelyk Template](#) (gtpl) we are going to only use [Groovlets](#) which are simple Groovy files

Create a folder called **groovy** in WEB-INF folder. All the groovy files in this folder will be groovlets.



routes.groovy - Gaelyk provides an excellent [URL routing](#) mechanism which provides for customized - REST - here is the basic routes.groovy for the application

```

all "/datastore", forward: "/datastore.groovy"
all "/blob", forward: "/blob.groovy"

```

Groovy/Gaelyk Side

Working with the Datastore (**Incomplete) - Datastore groovlet - WEB-INF/groovy/datastore.groovy.

```

import com.google.appengine.api.datastore.Entity
switch(params.a) {
    case "save": // Datastore Save action
        Entity entity = new Entity("person")
        // This is a shortcut Gaelyk "sugar" to retrieve all the params and
        save them as an Entity

```

```

    entity << params
    entity.save()
    sout << entity.name
    sout << entity.email
    sout << entity.profilephotokey
    break
case "delete":
    // NOT YET IMPLEMENTED
    break
case "update":
    // NOT YET IMPLEMENTED
    break
case "query":
    // NOT YET IMPLEMENTED
    break
}

```

Working with the Blobstore - Blob Groovlet - WEB-INF/groovy/blob.groovy

```

import com.google.appengine.api.blobstore.BlobKey
import com.google.appengine.api.blobstore.BlobstoreService
import com.google.appengine.api.blobstore.BlobstoreServiceFactory
// This is a simple blob.groovy file which has all the methods needed to
// upload and serve a blob.
// get the Params a (action)
switch(params.a) {
    case "geturl": // Getting the blobstore URL
        // Getting the
        sout << blobstore.createUploadUrl(params.url)
        break
    case "submit": // After submit
        // Getting the request
        def blobs = blobstore.getUploadedBlobs(request)
        // Name of the blob being sent through
        def blob = blobs["profilephoto"]
        // Setting the response code to 302 - redirect
        response.status = 302
        // If the blob is stored redirect to self - action success else
        failure - pass the key
}

```

```

        if (blob) {
            redirect "blob?a=success&key=${blob.keyString}"
        } else {
            redirect "blob?a=failure"
        }
        break
    case "success": // On Success
        // For REST like interface checking on GWT set the error code to
200
        response.status = 200
        // Output the key as a response
        out << params.key
        break
    case "failure": // On Failure
        response.status = 500
        out << "Failed to retrieve"
        break
    case "serv": // Serving the Blob (in this case Image
        // Get the Blob Key from params
        BlobKey blobKey = new BlobKey(params.key)
        // Get the Image from BlobStore
        def image = blobKey.image
        // Apply Transformation
        def thumbnail = images.applyTransform(images.makeResize(100, 200),
image)

        // Set the content type
        response.headers['Content-Type'] = blobKey.contentType
        // provider the thumbnail imagedata back as response
        sout << thumbnail.imageData
        break
}

```

GWT Side

Starting a new BlobSession and updating the FormPanel's action to point to the BlobStore's URL - Note the use of GWT's RequestBuilder - this is setting the Blobstore's unique URL as the action.

```

// Calling the startNewBlobSession
startNewBlobSession(formProfilePhoto);

```

[..]

```
private void startNewBlobSession(final FormPanel fp) {
    RequestBuilder rb = new RequestBuilder(RequestBuilder.POST, "/blob");
    rb.setHeader("Content-Type", "application/x-www-form-urlencoded");
    try {
        // Refer to the blob.groovy action 'geturl'. We are asking
        // Blobstore to redirect to the url /blob.groovy?a=submit
        Request request = rb.sendRequest(
            "a=geturl&url=/blob.groovy?a=submit",
            new RequestCallback() {
                public void onResponseReceived(Request request,
                    Response response) {
                    // On Successful response - set the FormPanel's action
                    fp.setAction(response.getText());
                }

                public void onError(Request request,
                    Throwable exception) {
                    Window.alert("Bugger - something went wrong");
                }
            });
    } catch (RequestException e) {
        e.printStackTrace();
    }
}
```

[..]

```
formProfilePhoto.addSubmitCompleteHandler(new FormPanel.SubmitCompleteHandler()
{
    public void onSubmitComplete(SubmitCompleteEvent event) {
        startNewBlobSession(formProfilePhoto);
        String key = event.getResults();
        profileBlobKey = key;
        Image img = new Image("/blob?a=serv&key=" + key);
        hpProfilePhoto.add(img);
    }
});
```

Saving to the Datastore is similar to above, much simpler.

```
RequestBuilder rb = new RequestBuilder(RequestBuilder.POST, "/datastore");
try {
    rb.setHeader("Content-Type",
        "application/x-www-form-urlencoded");
    Request request = rb.sendRequest(
        "a=save&name=" + tbName.getText()
        + "&profilephotokey=" + profileBlobKey
        + "&email=" + tbEmail.getText(),
        new RequestCallback() {
            public void onResponseReceived(Request request,
                Response response) {
                ft.setWidget(3, 0,
                    new Label(response.getText()));
            }

            public void onError(Request request,
                Throwable exception) {
                Window.alert("Bugger - something went
wrong");
            }
        });
} catch (RequestException e) {
    e.printStackTrace();
}
```

I have attached the sources - to this post. This tutorial will be updated in the near future.