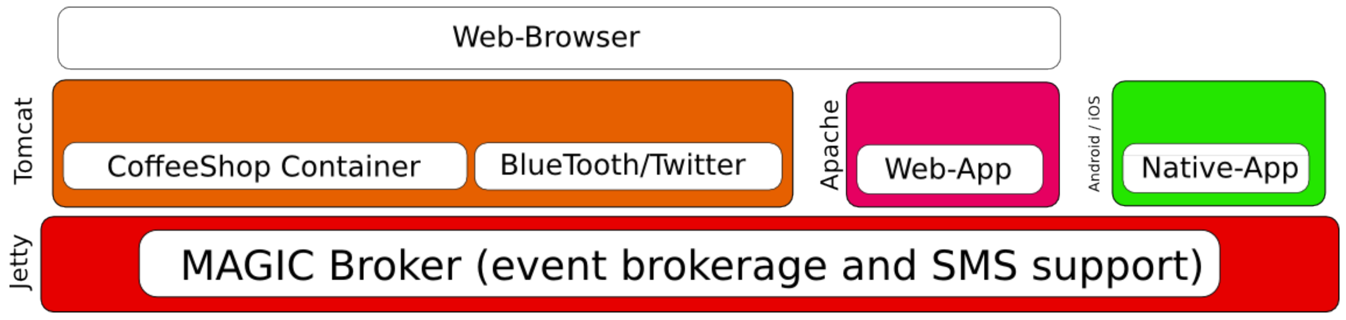
HOW TO CREATE A SIMPLE MOBILE WEB APPLICATION TO INTERACT WITH LARGE DISPLAYS

By Roberto Calderon
Media And Graphics Interdisciplinary Centre
University of British Columbia.
rvca@interchange.ubc.ca

We are going to create a very simple mobile web application that uses the MAGIC Broker to send/receive events and the CoffeeShop Display Framework to contain it.

The code is in the root of the CoffeeShop project, under the folder “counter”. The mobile-web application is under the “mobile” folder and the large display application is under the “large” folder. To deploy copy both folder to the “*/var/www/*” folder in an Ubuntu box, or under your public html folder for your apache2 server.

ABOUT THE ARCHITECTURE

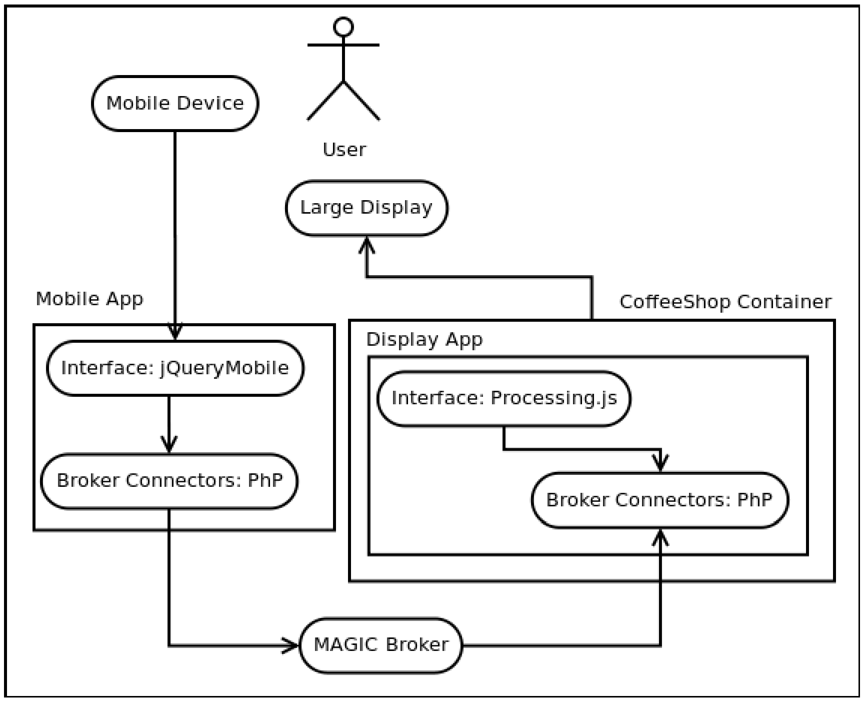


On the bottom of everything lives the MAGIC Broker. The Broker handles events as topics and allows for SMS to be easily configured. This is a self-contained application that can be run when downloaded.

We then run two servers, Tomcat and Apache. Apache is used to host web-based applications that can be “contained” in the CoffeeShop cointainer. Tomcat runs our CoffeeShop Framework that can be easily configured to run many applications in a large display. Note that currently each CoffeeShop Container can only run ONE single display. We are working on a multi-display container, but it's not ready for deployment. The role of the CoffeeShop container is to queue and de-queue applications so that a single display can be used to access up to 4 applications. Finally there is the possibility to write native mobile applications. All of these components depend on the MAGIC Broker.

ABOUT THE ARCHITECTURE OF OUR APP

We are going to separate the two parts of our application in a mobile-web application and a display application. The mobile-web application can be replaces with a native application to any mobile device (e.g. Android, iOS) but you will learn the main concepts of how to interact with a large display here. The large display application will be managed by the CoffeeShop container. It will look like this:



The user uses his mobile device to access a JQueryMobile website hosted in an apache server. This application has a javascript function that will call a PhP script when actions are taken by the user (e.g. Swiping, geo-data, accelerometer, etc). The PhP file is a dirty but very useful trick to bypass javascripts Access-Control locks that prevents calls to other domains (an app visited at http://localhost cannot make ajax calls to http://call.me) The PhP script will make a GET request to the MAGIC Broker Rest API to either send events or get events to a particular topic.

NOTE: Topics are similar to channels, so for our application we are going to create a “counter” channel where both our “counter_mobile” and our “counter_display” clients will post and receive events. Events are name-value pairs of information, an event can have as many name-value pairs as you want.

The broker will then store those events until someone requests them. If you request the events they will disappear from the queue.

The display application uses Processing.js (processing implementation for javascript) to visualize events. A function called every second calls a PhP script to look for events posted to the “counter” topic. The display application is then contained within the CoffeeShop Container.

ABOUT THE MAGIC BROKER API

To use the MAGIC Broker you need to do the following:

1) Subscribe a client to a topic. If the topic doesn't exist it gets created.
2) A client can send events to a topic.
3) Any other client, or the same, can query for events in a topic.
4) If you unsubscribe all clients from a topic, the topic gets deleted.

The MAGIC Broker can be accessed through the REST API and documentation for doing this can be found here:

http://www.magic.ubc.ca/wiki/pmwiki.php/OSGiBroker/Osgibroker-rest

From now on we will not explain the REST calls further, but you can find about them above.

THE DISPLAY APPLICATION

We will create a php script o subscribe our application to the "counter" topic:

```
<?php
$daurl = 'http://localhost:8800/osgibroker/subscribe?topic=counter&clientID=counter_large';

// Get that website's content
$handle = fopen($daurl, "r");

// If there is something, read and return
if ($handle) {
    while (!feof($handle)) {
        $buffer = fgets($handle, 4096);
        echo $buffer;
    }
    fclose($handle);
}
?>
```

This script calls the broker REST api and subscribes the counter_large client to the counter topic. If there is a response it will return it. Save it as subscribe.php.

We will now create a php script to query a topic for incoming events, we will time out our request after 1 second. For most applications a latency of 1 second is enough. For example, accelerometer data needs to be smoothed and interpolated. We write our getevent.php file:

```
<?php
//The minimum timeout allowed by the broker's rest API is 1 second. For most large-display-to-mobile
//applications a second latency is enough (e.g. accelerometer data needs to be interpolated and averaged)
$daurl = 'http://localhost:8800/osgibroker/event?topic=counter&clientID=counter_large&timeOut=1';

// Get that website's content
$handle = fopen($daurl, "r");

// If there is something, read and return
if ($handle) {
    while (!feof($handle)) {
        $buffer = fgets($handle, 4096);
        echo $buffer;
    }
    fclose($handle);
}
?>
```

Now, download the processing.js library you can find at http://processingjs.org/ Now, let's write our html/javascript file. Let's create our index.html file:

```
<html>
<head>
<script type='text/javascript' src='http://code.jquery.com/jquery-1.6.2.js'></script>
<script src="processing.js"></script>

<script>
    //Declare a global value to use later in a processing.js script
    var rotation = 1;

    $(document).ready(function(){
       //subscribe the large display application to the clock topic
       $.ajax({ type: "GET", url: "subscribe.php"});

       //retreive events every second and parse them.
       setInterval(function () {
         $.ajax({ type: "GET", url: "getevent.php", dataType: "xml", success: messageParser });
       }, 1500);
    });

    function messageParser(xml) {
      //for each event received by the broker we will add 5 "seconds" to our counter.
      $(xml).find("event").each(function () {
        rotation = rotation + 5;
        //we are doing a "clock" so only allow rotation to be up to 60 "seconds"
        if (rotation > 61) {
                rotation = 1;
              }
      });
    }
</script>

</head>
<body>
<h2>Simple Large Display Interaction.</h2>
<p>Add 5 seconds to our "counter" for each swipe on our web-mobile app.</p>
<p><canvas id="canvas1" width="500" height="500"></canvas></p>

<script id="script1" type="text/javascript">

// Simple way to attach js code to the canvas is by using a function
function sketchProc(processing) {
  // Override draw function, by default it will be called 60 times per second
  processing.draw = function() {
    // determine center and max clock arm length
    var centerX = processing.width / 2, centerY = processing.height / 2;
    var maxArmLength = Math.min(centerX, centerY);
    function drawArm(position, lengthScale, weight) {
      processing.strokeWeight(weight);
      processing.line(centerX, centerY,
        centerX + Math.sin(position * 2 * Math.PI) * lengthScale * maxArmLength,
        centerY - Math.cos(position * 2 * Math.PI) * lengthScale * maxArmLength);
    }

    // erase background
    processing.background(224);

    // Moving hour arm by second increments
    var secondsPosition = rotation/60;
    drawArm(secondsPosition, 0.90, 1);
  };

}

var canvas = document.getElementById("canvas1");
// attaching the sketchProc function to the canvas
var p = new Processing(canvas, sketchProc);
</script>

</body>
</html>
```

As you can see we use jquery to manage make ajax calls to our php scripts hosted locally. We use processing.js to visualize our information. You can find more information for both jquery and processing.js at:

http://processingjs.org/
http://jquery.com/

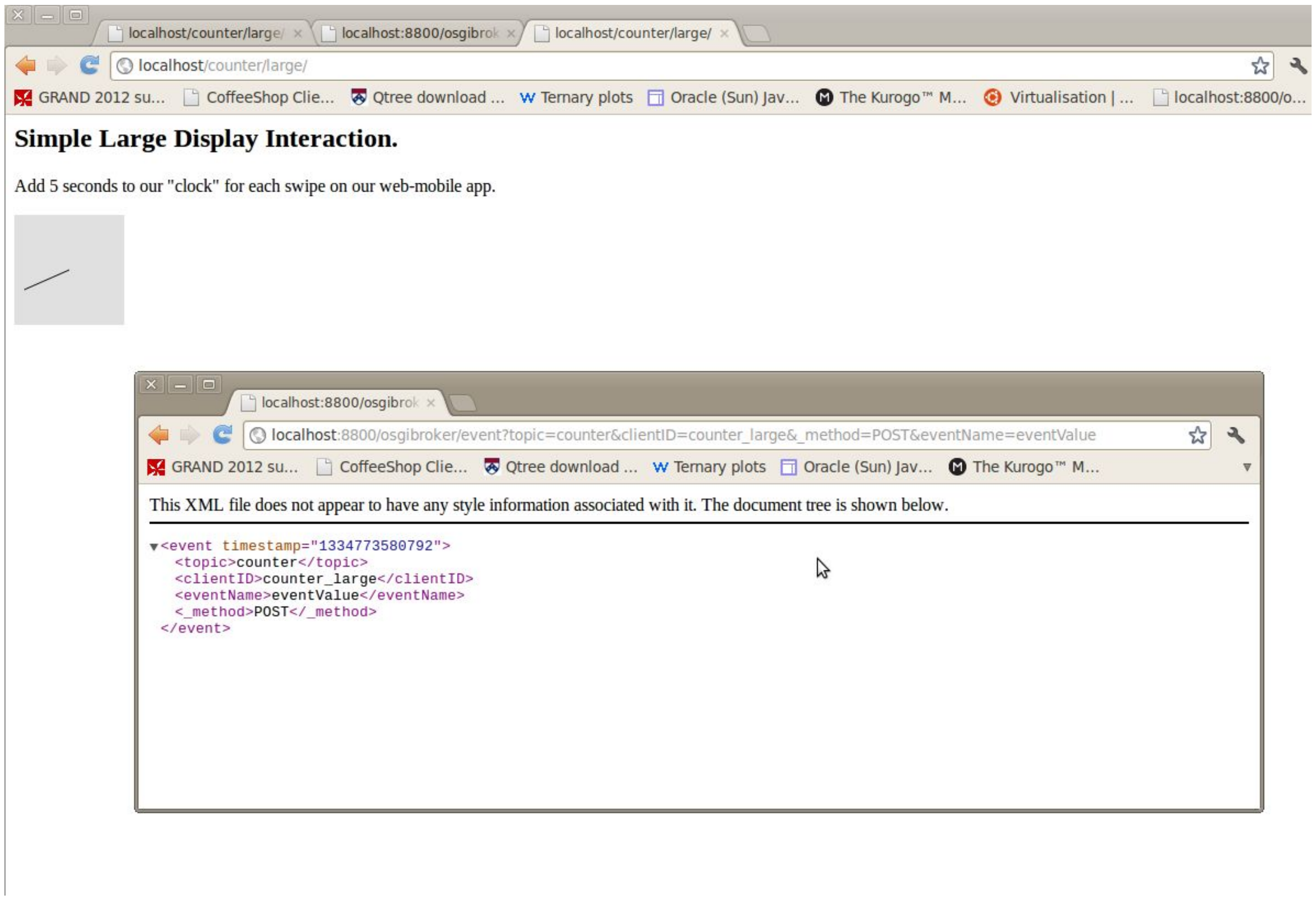
Visit http://localhost/counter/large/index.html

Wow, that's kind of cool!! But hey, it doesn't do anything... Let's send an event to the Broker. Open another web-browser window and hit the url:

http://localhost:8800/osgibroker/event?topic=counter&clientID=counter_mobile&_method=POST&eventName=eventValue

Every time you hit this URL our counter will go up for “5 seconds”.

This url is only sending a dummy name-value pair. If you recall our index.html parse function, as long as we get an event we will visualize it. You can however parse for that event and visualize anything. It could be coordinates, accelerometer direction and value. Anything!



Simple Large Display Interaction.

Add 5 seconds to our "clock" for each swipe on our web-mobile app.

localhost:8800/osgibroker/event?topic=counter&clientID=counter_large&_method=POST&eventName=eventValue

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▾<event timestamp="1334773580792">
   <topic>counter</topic>
   <clientID>counter_large</clientID>
   <eventName>eventValue</eventName>
   <_method>POST</_method>
 </event>
```

THE MOBILE-WEB APPLICATION

On a different project, but also hosted by an Apache server, we will create a new application. This application will send events to the MAGIC Broker to the “counter” topic, as you recall these events will be then read by our large display application.

So let's write our subscribe.php script. Yes, it's a different app, and client, so we need to subscribe it to our “counter” topic:

```
<?php
$daurl = 'http://localhost:8800/osgibroker/subscribe?topic=counter&clientID=counter_mobile';

// Get that website's content
$handle = fopen($daurl, "r");

// If there is something, read and return
if ($handle) {
    while (!feof($handle)) {
        $buffer = fgets($handle, 4096);
        echo $buffer;
    }
    fclose($handle);
}
?>
```

Now, let's write our sendevent.php script. This will send a dummy name-value pair.

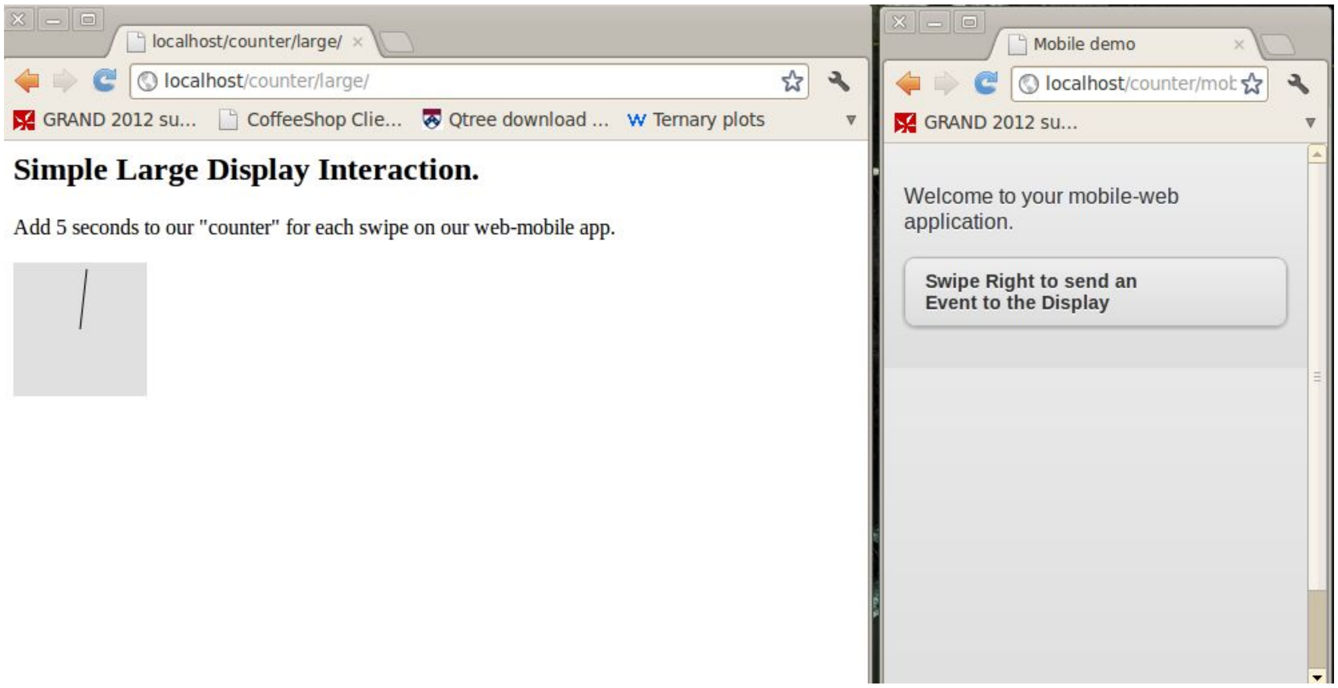
```
<?php
// We are going to send a dummy name-value pair. If you are sending discrete data (e.g. accelerometer data)
// you would send an event containing a direction, or a coordinate.
$daurl = 'http://localhost:8800/osgibroker/event?
topic=counter&clientID=counter_mobile&_method=POST&eventName=eventValue';

// Get that website's content
$handle = fopen($daurl, "r");

// If there is something, read and return
if ($handle) {
    while (!feof($handle)) {
        $buffer = fgets($handle, 4096);
        echo $buffer;
    }
    fclose($handle);
}
?>
```

It's time for the fun part. We are going to use JqueryMobile to create a super-fast demo . You can find all the documentation and download the code and examples at http://jquerymobile.com/. Our application will use of the JqueryMobile swipe event to make an ajax call to our sendevent.php script, in order to send a dummy event to the MAGIC Broker. We will also show some feedback telling our user that we have sent that event. This is quite basic, but it shows the basic structure of a mobile app. You can experiment with html5's capabilities for more mobile-based data. You can also write a native application that can get accelerometer data, private information, or other fancy stuff. The concept is the same, you create an application that makes a RESTFUL call to the Broker to post data that will later be read by a large diaplay application. Here's the code for the mobile app:

```
<!DOCTYPE html>
<html>
<head>

  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title> Mobile demo</title>
  <script type='text/javascript' src='http://code.jquery.com/jquery-1.6.2.js'></script>
  <script type="text/javascript" src="http://code.jquery.com/mobile/1.0b2/jquery.mobile-1.0b2.js"></script>
  <link rel="stylesheet" type="text/css" href="http://code.jquery.com/mobile/1.0b2/jquery.mobile-1.0b2.min.css">

  <style type='text/css'>
  </style>

  <script type='text/javascript'>
    $(window).load(function(){
      $("#listitem").swiperight(function() {
        $.mobile.changePage("#page1");
        $.ajax({ type: "GET", url: "sendevent.php" });
      });
    });

</script>

</head>
```

```
<body>
  <div data-role="page" id="home">
    <div data-role="content">

          <p>Welcome to your mobile-web application.</p>
        <p>
          <ul data-role="listview" data-inset="true" data-theme="c">
            <li id="listitem">Swipe Right to send an Event to the Display</li>
          </ul>
        </p>
    </div>
</div>

<div data-role="page" id="page1">
    <div data-role="content">

        <ul data-role="listview" data-inset="true" data-theme="c">
          <li data-role="list-divider">Congratulations</li>
          <li><a href="#home">Back to the Home Page</a></li>
        </ul>

        <p>
          YAY!<br/>You Swiped Right and sent an event.
        </p>
    </div>
</div>

</body>


</html>
```

Now visit http://localhost/counter/mobile/index.html and play with it. Now that's cool.



Simple Large Display Interaction.

Add 5 seconds to our "counter" for each swipe on our web-mobile app.

Welcome to your mobile-web application.

Swipe Right to send an Event to the Display

CONFIGURING THE COFFEESHOP CONTAINER

Time to get all things done. If you haven't done so, go to the Getting Started Guide found where you found this file (the CoffeeShop project under the folder “documentation”).

We are going to create two files a counter.xml configuration file and a CounterConnector.java file. The former will configure where the large display application lives and the name, as well as the connector use by the CoffeeShop cointainer to queue and de-queue the application. The CounterConnector.java implements the Connector interface and declares the name of the topics to keep an eye on.

Open Eclipse and import the CoffeeShop project. Under the package *ca.ubc.magic.coffeeshop.config* create a counter.xml file to declare where the application is accessible (http://localhost/counter/large/index.html), where the thumbnail lives and what connector to use:

```
<?xml version="1.0" encoding="UTF-8"?>
<pluginApp xmlns="http://pspi.magic.ubc.ca"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://pspi.magic.ubc.ca application.xsd">

    <applicationName>Counter</applicationName>
    <applicationShortName>counter</applicationShortName>
    <applicationImageURL>http://localhost/counter/counter.png</applicationImageURL>
    <minumumIdleTime>20</minumumIdleTime>

    <applicationDescription>This application is a sample application.</applicationDescription>
    <interactionInstructions>Visit the Large-Mobile guide.</interactionInstructions>

    <connectionInfo applicationType="web" useFullScreen="false">
        <topic>hello</topic>
        <displayURL>http://localhost/counter/large/index.html</displayURL>
        <connectorClass>ca.ubc.magic.coffeeshop.connectors.CounterConnector</connectorClass>
    </connectionInfo>

</pluginApp>
```

Now, under the package *ca.ubc.magic.coffeeshop[.connectors*, *c*reate a CounterConnector.java file and implement the connector interface. Otherwise you can just copy the HelloConnector.java file and update the OSGI broker topic and client accordingly to match the one used by your application (i.e. the one that CoffeeShop will keep an eye on):

```
package ca.ubc.magic.coffeeshop.connectors;

import java.io.IOException;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.UnknownHostException;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import ca.ubc.magic.coffeeshop.classes.CoffeeShop;
import ca.ubc.magic.coffeeshop.jaxb.Application;
import ca.ubc.magic.osgibroker.OSGiBrokerClient;
import ca.ubc.magic.osgibroker.OSGiBrokerException;
import ca.ubc.magic.osgibroker.OSGiBrokerService;
import ca.ubc.magic.osgibroker.workgroups.TopicEvent;

public class CounterConnector implements Connector {

        CoffeeShop cs;
        String topic = "counter";
        String topicConnector = "cs_counter";

        OSGiBrokerService BrokerCounter;
        OSGiBrokerClient ClientCounter;

        public CounterConnector() throws OSGiBrokerException {

                try {
                        cs = CoffeeShop.getInstance();

                        BrokerCounter = new OSGiBrokerService("localhost:8800");
                        ClientCounter = BrokerCounter.addClient("counterconnector");

                        try {
                                ClientCounter.subscriber().subscribeHttp("counter", false);
                                ClientCounter.subscriber().subscribeHttp("cs_counter", false);

                        } catch (OSGiBrokerException e1) {
                                e1.printStackTrace();
                        }

                        Map<String, String> map = new HashMap<String, String>();
                        map.put("message", "CONSTRUCTOR");
                        sendEvent(map);

                        new Thread(new SocialwallListener()).start();

                }
                catch (UnknownHostException e) {
                        e.printStackTrace();
                }
                catch (IOException e) {
                        e.printStackTrace();
                }



        }

        class SocialwallListener implements Runnable {

                @Override
                public void run() {

                        while (true){
                                try {
                                        TopicEvent[] events = ClientCounter.subscriber().getEvents("counter", 1);

                                        if ( events.length > 0 ) { // if we have at least one event.
                                                Map<String, String> map = new HashMap<String, String>();
                                                map.put("message", "RUNNINGSTILL" );
                                                ClientCounter.publisher().sendEvent("cs_counter", map);

                                        }

                                        Thread.sleep(5000);

                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }
                                catch (OSGiBrokerException e1) {
                                        e1.printStackTrace();
                                }
                        }

                }
```

```
	}

	@Override
	public void receiveEvent(TopicEvent event) {
		// Nothing to do here. No need to do anything on receive.
	}

	@Override
	public void sendEvent(Map<String, String> paramaters) {
		//
	}


}
```

You can probably figure out what to change for another application by reading the code. Re-compile the CoffeeShop container and upload the war file to your Tomcat server, restart the server and you're done!