|  |
| --- |
| UBC MAGIC |
| Coffee Shop Container Application |
| User Guide |
|  |
|  |
|  |
| **3/8/2010** |

**Rustin Plowright-Wills**

**Jay Wakefield**

|  |
| --- |
|  |

Table of Contents

[List of Figures 3](#_Toc259560097)

[Purpose 4](#_Toc259560098)

[Public User Guide 5](#_Toc259560099)

[Interacting via HTTP 5](#_Toc259560100)

[Interacting via SMS 7](#_Toc259560101)

[Administrator User Guide 9](#_Toc259560102)

[External Application Programmer Guide 12](#_Toc259560103)

[Adding Applications via XML Definition Files 12](#_Toc259560104)

[Adding Applications via Web Interface 15](#_Toc259560105)

[Maintenance Guide 17](#_Toc259560106)

[Container Code Base Maintenance 17](#_Toc259560107)

[Building and Deployment 17](#_Toc259560108)

# List of Figures

[Figure 1 - User Selection Page 5](#_Toc259560109)

[Figure 2 - Queuing an Application 6](#_Toc259560110)

[Figure 3 - Administration Page 9](#_Toc259560111)

[Figure 4 - Application Editing 10](#_Toc259560112)

[Figure 5 - Sample XML Definition File 13](#_Toc259560113)

[Figure 6 - New Application Page 16](#_Toc259560114)

# Purpose

This document is being created to provide a multipurpose guide for using the container application. It provides a guide on how to use the application, but does not provide a technical overview of the project. It only specifically provides information that a particular type of user needs. It includes four main sections – a public user guide, administrator user guide, an external application programmer guide, and maintenance guide. The important aspects the container application to each of these specific roles are outlined in the sections to follow.

For technical documentation, including application design and structure, overview of protocols, and implementation, please refer to the technical documentation provided for the project.

# Public User Guide

This section of the user guide provides information regarding how to interact with the container from a public perspective. This includes information on how to select applications, write to the message board and interact with the currently running application.

## Interacting via HTTP

The container application provides a web interface for public users to interact with the large display. This page allows users to select applications to run, send messages to the message board and provides instructions on how to interact with the currently running application. An example of how this page looks is below in figure 1.

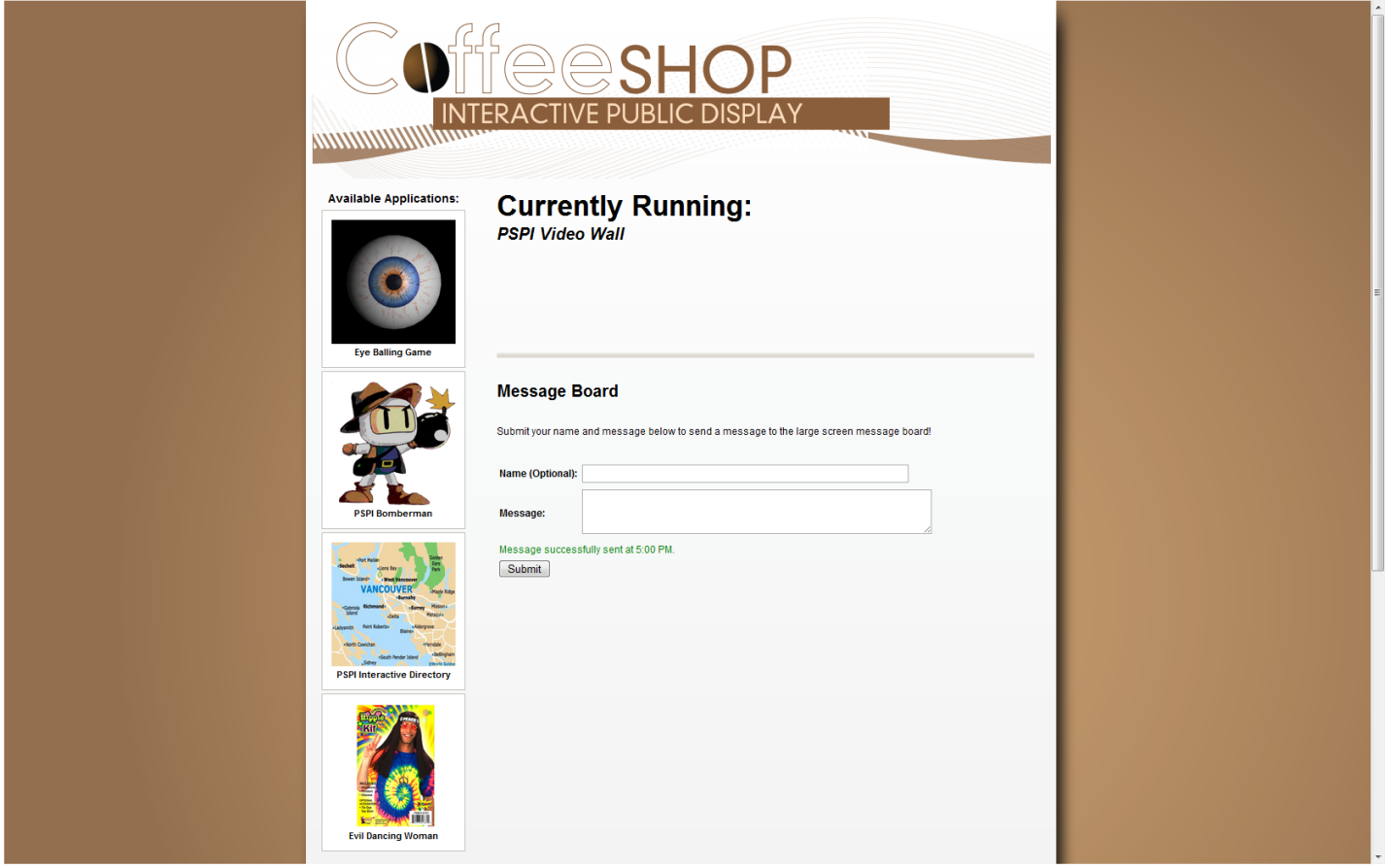


Figure - User Selection Page

In the main part of the screen, the page displays information about the currently running application. This includes information like the name of the application, the description, and any instructions on how to interact with it. All of these items are displayed as configured by the application programmers and administrators. This section of the page is automatically updated to reflect what application is currently running on the large display.

On the left side of the screen, there are buttons for each of the applications that are configured to run in the container. If a user clicks on one of these buttons, the main part of the page updates with information about the selected application. The user then can select “Queue this application to run,” which will queue the application in the large display, or “cancel,” which will return the user to the description of the currently running application. An example of this is shown in figure 2 below.



Figure - Queuing an Application

On the bottom part of the screen, there is a form which allows users to enter a message to be posted to the message board. When they submit this message, a message will be displayed indicating whether or not the message was successfully sent. If the message was sent, the user will be able to see their message on the main screen.

The URL for this user interaction page will depend where the application is deployed, but will have general URL of: http://<web address>/CoffeeShop/client/select.jsf. The URL should also be displayed on the main user interface. However, not that that at the time of writing this document, display of the URL on the main screen had not yet been implemented.

## Interacting via SMS

Public users are also able to interact with the container via SMS (text) message. The phone number for which to send messages should be shown on the large display.

To select an application to run, the user should send a message starting with the word “select” followed by the number of the application as seen on the main screen. For example, if the user wishes to queue the second application displayed on the main screen to run, they would send “select 2” to the phone number provided. When the message is received, they should be able to see their application in the ready queue, which is located at the top of the main screen.

To send a message to the message board, the user should send a message starting with the word “say” followed by their message to the phone number provided. For example, if the user wanted to say “coffee is cool,” they should send the message “say coffee is cool” to the phone number provided. When the message is received, they will see their message posted to the message board on the large display.

Any message sent to the provided number that does not include either of the above keywords will be forwarded to the currently running application. The application may provide instructions on how to format a message to be sent to it. This is up to the application developer to provide. However, this is how one would interact with the currently running application via text message.

# Administrator User Guide

The container application provides an interface intended for administrators of the coffee shop application to quickly edit the configuration or state of applications. An example of the main configuration interface is seen below in figure 3.

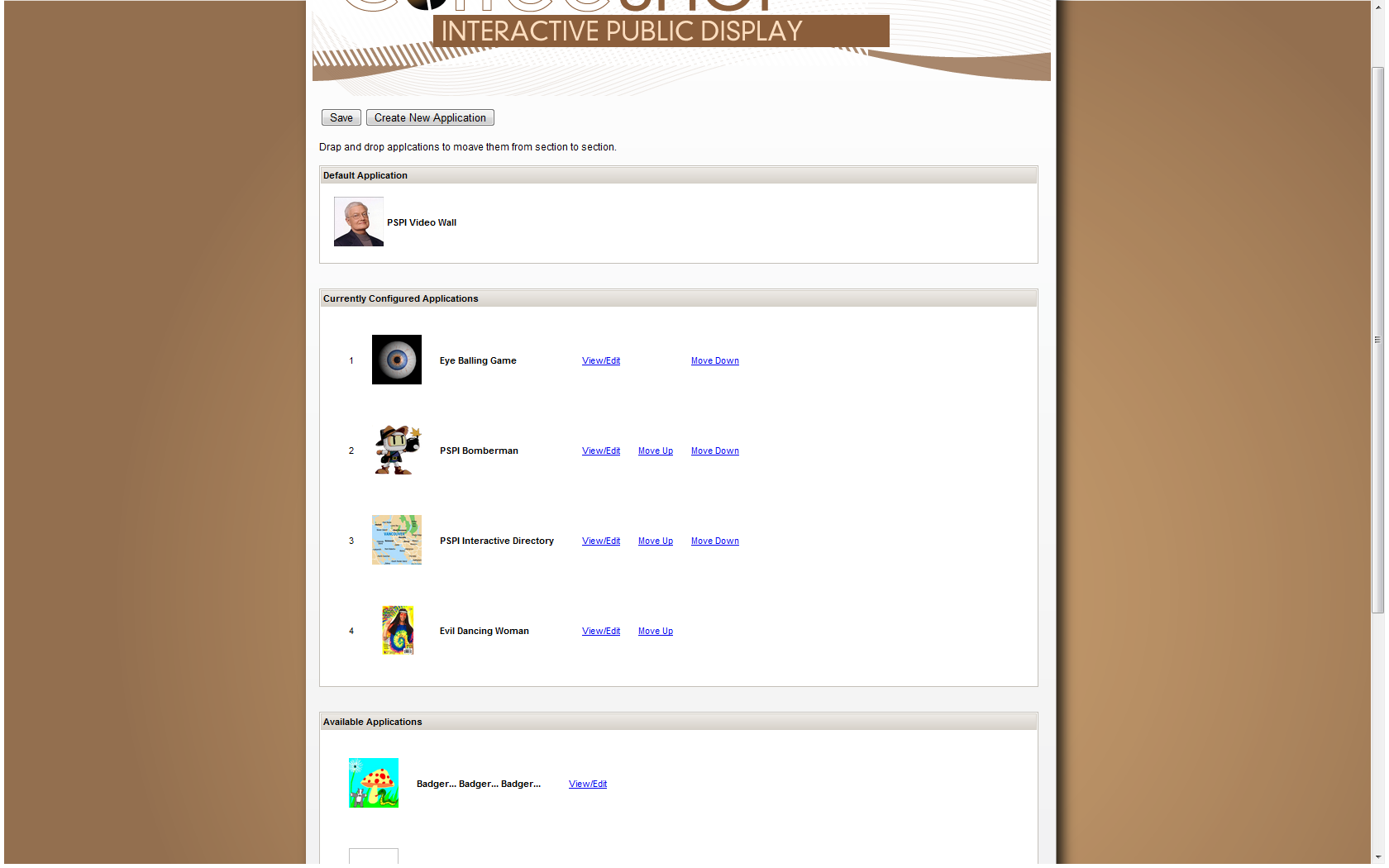


Figure - Administration Page

As seen in figure 3, there are three main sections to this page. These include a section for what application is currently configured as the default application, as section containing applications that are configured to be shown on the large display, and a section containing applications that are not configured to be run in the large display, but are available. The administrator may click and drag applications from section to section, and save the page to reconfigure the application. As they are dragging, there is a drag indicator which tells them whether the application will allow them to drop the application in the given section. For example, the administrator may drag from the all available section to the configured section, but may not drag directly to the default application section. This is evident with the drag indicator.

The administrator should be aware that until they save the page, no configuration changes have been made. Once saved, a message will be sent to the large display and user input page, and they will be updated accordingly.

The order of applications as listed on the large display is also configurable. To change the order, the administrator may select the “move up” and “move down” buttons next to the application name and image. Once saved, the order will be reflected on the large display and user input page.

The administrator is also able to edit the settings of a particular application dynamically. To do this, they would click the “view/edit” button next to their chosen application, and would be directed to the page seen in figure 4 below.



Figure - Application Editing

Here, the administrator may edit any details of the application, and select save. The changes will then be reflected on the large display and user input page. If the administrator chooses to cancel, no changes will have been made.

Finally, the administrator may choose to create a new application from scratch. From the main configuration page, they would click the button labeled “Create New Application.” They would then be forwarded to a blank page similar to the application view/edit page. They must fill in all of the required information, and select save. Once successfully saved, the new application will be listed in the “available applications” section of the main configuration page. At this point, the newly created application is available to be configured, and the administrator can use it the same way as any other application.

# External Application Programmer Guide

This section of the user guide provides information for an application programmer to manually configure the container application to run an external application. This section is mainly of interest to developers who have created their own application that they would like to run within the container application. It outlines two methods to configure applications – XML definition and web definition.

## Adding Applications via XML Definition Files

Adding applications to the container is done by having the user create an XML file for each additional application they wish to add provided that the application mechanisms are relatively simple. The XML file requires certain configuration information that allows the container to properly configure it for runtime.

Note that if there is anything particular that the container should do before or after an event is sent to the OSGiBroker, this method is preferred for configuration, as the programmer must also write a connector class. This is explained later in this section.

To configure a new application via XML, the programmer must create a new xml file with a one word name. This name is the “short name” of the application, and will be used in various places for configuration. For example, if the name of my application is “Driving Tricycles,” I may use the short name “tricyles.” In this case, my XML file would have the name “tricycles.xml.” The programmer must place this completed file in the classpath of the application, in the package “ca.ubc.magic.coffeeshop.config.”

An example configuration file is shown in figure 5 below.

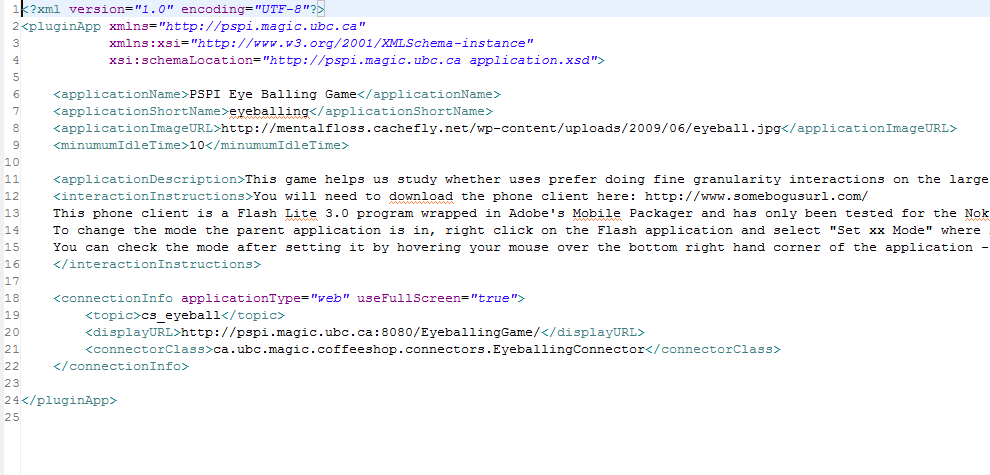


Figure - Sample XML Definition File

The following elements are used for configuration. All fields are required unless noted below:

* **Name:** The name of the application to be displayed.
* **Short Name:** A one word name to be used with communication and configuration. This is also the name of the defined xml file.
* **Image URL:** The URL of the thumbnail image to be associated with the application
* **Minimum Idle Time:** The amount of time, in seconds, in which the application should wait for interaction. If there is no interaction after the expiration of this time, the application running is switched.
* **Description (optional):** A general description of the application.
* **Instructions (optional):** A description of how to interact with the application when it is running.
* **Connection Information:**
  + **Topic:** The name of the topic for the OSGiBroker that the application should subscribe to when it is running.
  + **Display URL:** The URL of the application that should be displayed on the large screen when the application is running.
  + **Connector Class:** If the application requires a custom connector this element must define the class name to use. See below for more information.
  + **Application Type:** The type of the application. Either SWF (flash) or Web (typical web application).
  + **Use Full Screen:** This is a true or false value defining whether the application should run in full screen mode.
* **Parameters (optional):** This allows the user to define HTML form elements to be displayed on the HTML input page for interaction with the application. This includes text boxes, text areas, passwords, check boxes, drop down lists, and radio buttons. The programmer may also define the order that they appear on the page. The implementation of this does not yet exist, however it can easily be included given the information that is defined in the XML file.

Once the XML file has been created the user then has to configure the application to run by editing the container configuration in the config.properties file, which is found in the same package on the classpath of the application. The container application reads from this file to determine the details of connection, and which applications to load at runtime. To configure a new application to run there are three fields in the configuration file that the user must know about:

* **allApplications:** Contains a list of all possible applications to run. Each must have an XML file as described above.
* **applicationsToRun:** Contains a subset of the applications in *allApplications* that have been selected to run in the container. Each must have an XML file as described above and be in the *allApplications* list. The applications will appear in the container application in the order that they are defined in *applicationsToRun* with the exception of *defaultApplication*.
* **defaultApplication:** The default application to run when there are no requests from users. *defaultApplication* must also be listed in *applicationsToRun* but will not appear in the visual list of applications defined by *applicationsToRun*.

In some cases it is required that the application to be added to the container requires more interaction than the information provided in the XML file. In this case the user may create a new Java class that specifies what happens when the application sends or receives an event.

A Java interface, *Connector*, is defined to allow a programmer to create their connector class and must implement the following methods:

* **public void sendEvent(Map<String, String> params);**
  + This method is invoked when a user interacts with the application, either via HTTP or SMS protocol, and contains the messages that the user sent along with their interaction. The programmer would need to write an implementation of what they would like to do before the event is sent to the application via the OSGiBroker. Note that the programmer **MUST** also send an event to the OSGiBroker to inform it that an event has occurred. This can be done by using the CoffeeShop API.
* **public void recieveEvent(TopicEvent e);**
  + This method is invoked when the application receives an event from the OSGiBroker and contains the information the OSGiBroker has sent to it in the TopicEvent object. The programmer would need to write an implementation of what they would like to do on the receipt of an event from their application.

All of the application XML files and the config.properties file must be available in the classpath in the package “ca.ubc.magic.coffeeshop.config.” If the files are not in this package, they will not be found by the application class loader, and the container will fail. As well, all custom connector classes must be created in the “ca.ubc.magic.coffeeshop.connectors” package for similar reasons.

## Adding Applications via Web Interface

When the container is running, programmers have the ability to configure and add a new application via web form, assuming there is no need for a custom connector class. In this case, the user would navigate to the administration page, as seen in figure 3, and would select “Create New Application.” They would then be forwarded to the page seen below in figure 6.



Figure - New Application Page

The use would need to fill in the appropriate fields on the page to define their new application. These fields have the same meanings as those in the XML file, so the previous section can be used for reference when filling out the form.

Once the application is created and saved, it will be visible in the “available application” section of the administration page, and can be figured as described in the “Administrator User Guide” section.

# Maintenance Guide

This section is for administrators to be able to quickly find the files needed for configuration and maintenance. It includes information on how to build the application, and what properties to configure when the application is deployed.

## Container Code Base Maintenance

The current location of the code for the container application is on the PSPI subversion repository “creole.” You will probably need to ask someone in the magic lab for rights to access this repository. Once you have access to the code, detailed descriptions of the structure and design of the code can be found in both the technical and non-technical report for this project. They should be available in the same place that this document is available, but if it is not, ask the appropriate member of the PSPI project for help.

The original project has a configured set up to be used with the Eclipse IDE and Tomcat Application Server 6.0. It is so be used as a Web Application Project.

## Building and Deployment

Ant build scripts have been created to assist users to build and deploy the container application. In order for the build scripts to work properly, a JDK and Flex SDK must be installed on the system you wish to build on, and the JAVA\_HOME and FLEX\_HOME environment files must point to the appropriate locations. Also, both the server side project “CoffeeShop” and the front end project “guiFramework” must be located in the same workspace.

The entire project can be built from the ant script in the “CoffeeShop” server side project. The ant script has the following build targets:

* **Init**: This target simply creates build folders and locations in preparation for the build.
* **Compile**: This target compiles the java source files for the project, and moves them to the build location created in the init target. It also moves configuration files that must be in the classpath to the same location, in the appropriate directory structure.
* **Inject-gui**: This target builds and injects the GUI flash file from the guiFramework. Once built, it moves the file into the appropriate directory of the server side project. This target can be used for development work as well, as it will build the version of the GUI in your workspace, and move it into the web project which can then be deployed locally for testing.
* **Build**: This target performs all of the above tasks and creates a new WAR file for deployment. This file is put in the “dist” folder of the project. This new WAR file is now ready to be deployed on any application server that supports java web applications.
* **Clean**: This target removes all compiled java files from the workspace.
* **Cleanall**: This target removes all compiled java files from the workspace, and removes the “dist” folder and its contents.

The default target for this ant script is “build.” If you wish to run other build targets, consult the documentation on running ant alone, or with eclipse.