Client Logic for Firefox Requirements Specification

1. Scope

1.1 Overview

The Client Logic for Firefox component provides client-side logic supporting Mozilla Firefox users' interaction with the Orpheus application. This client logic will be incorporated into a Firefox extension that enables Firefox to be used as an Orpheus client.

1.2 Logic Requirements

1.2.1 General Strategy

The overall Orpheus client will take the form of a browser extension whose user interface manifests through a browser tool bar, popup browser windows, and items added to the context menu. It will periodically poll the server for updates, and will rely on HTML pages loaded from the server for most of the content it presents. The Orpheus Client Logic for Firefox component will manifest as a Java class (or classes) that

- provides appropriate handling for UI events generated by user interaction with the client-side controls,
- monitors browser / document events to issue alerts to the user at appropriate times,
- periodically polls the Orpheus server for application state updates,
- exposes specific methods for interaction with scripts,
- provides an event listener interface by which it can notify the rest of its plugin about events, and
- as needed, pops up new browser windows in which to display information to or solicit information from the user.

The specifics of these behaviors are described in the remaining requirements.

1.2.2 Maintain a Bloom Filter Instance

The component will maintain a Bloom filter instance with which it will test the domains of sites visited by the user to determine whether they are participating in the Orpheus application. (See requirement 1.2.4.1.) This filter is initially empty, and will periodically be updated via messages pulled from the Orpheus server. It does not need to persist beyond the lifetime of one component instance.

1.2.3 UI Event Handlers

The component will provide event handlers intended to directly support various user interface controls. In all cases these handlers will cause HTTP requests to be issued to the server, where the specific URL to use in each case will be part of the component configuration. Most of the handlers open or reuse pop up browser windows (see requirement 1.2.7) to display the HTTP response; when this happens the new window will have minimal controls and adornment – ideally only a frame with title and close button.

1.2.3.1 Log In

The component will provide support for a "Log In" button that displays a login page from the game server. Upon successful login via the provided page, the component will fire a corresponding event to registered listeners (see requirement 1.2.6.4).

1.2.3.2 Log Out

The component will provide support for a “Log Out” button that issues an HTTP GET request to the configured log out page without using any popup window. Upon successful logout the component will fire a corresponding event to registered listeners (see requirement 1.2.6.4).

1.2.3.3 Show Active Games

The component will provide support for a “Show Active Games” button that displays a corresponding page from the game server.

1.2.3.4 Show User’s Games

The component will provide support for a “Show My Games” button that displays a corresponding page from the game server.

1.2.3.5 Show Unlocked Domains

The component will provide support for a “Show Unlocked Domains” button that displays a corresponding page from the game server.

1.2.3.6 Show Upcoming Domains

The component will provide support for a “Show Upcoming Games” button that displays a corresponding page from the game server.

1.2.3.7 Show Leaders

The component will provide support for a “Show Leaders” button that displays a corresponding page from the game server.

1.2.3.8 Show Latest Clue

The component will provide support for a “Show Latest Clue” button that displays a corresponding page from the game server.

1.2.4 General Event Handling

The component will provide event handlers intended to support the following scenario not directly responsive to UI controls:

1.2.4.1 Displayed Page Changed

The component will support testing the URLs of pages displayed in the browser. When a new page is displayed in the browser window (active tab) or when the active tab is changed, the host browser extension will generate events that this method (or methods) is intended to handle. In response, the component will

- determine whether the domain is a host site in any current game by extracting the domain portion (host name) of the page URL and testing for its presence in the current Bloom filter. If found then the component will
- query the server to determine for which games (if any) the domain is currently a host, relative to the player’s progress in any games for which he is registered. The domain will be specified in the request URL as a query parameter of configurable name. The response will consist of a single decimal number representing the number of active games for which the domain is a host on which the player has yet to complete his task. If the number is nonzero then the component will
- open or reuse a popup window to display the response to an HTTP request issued to a configurable URL on the Orpheus server, in much the same way described in

requirement 1.2.3. The domain will be specified in the request URL as a query parameter of the same configurable name used for the previous request.

1.2.5 Periodic Polling

The component will periodically poll for updates at a configured URL on the Orpheus server. The response will be interpreted as an Atom 1.0 feed document, with each entry representing a distinct message to this component or for display to the user. The nature of each entry is determined by the type attribute of its <content> element:

- if “text”, “html”, or “xhtml” then the content is a message – possibly marked up – to be presented to the user in a popup window of the kind described in requirement 1.2.7.
- The only other type supported in this version is “application/x-tc-bloom-filter”. The content assigned this type is expected to be a serialized Bloom Filter in the common format supported by TopCoder’s Java and .NET Bloom Filter components.

The component will retain the update timestamp of the feed, persisting it across browser shutdowns, and providing it back to the server with all feed requests as the value of a query parameter of configurable name.

The polling interval will be configurable, expressed as minutes elapsed between completion of one poll and initiation of the next.

1.2.6 Scripting Interface

1.2.6.1 Set / Get Working Game

The component will provide methods, intended to be exposed for scripting, by which a web page can specify or retrieve the numeric (64-bit signed integer) ID of a game to set as the current ‘working’ game. The component will persist this value locally, across browser restarts, until it is changed again via this interface. Upon change of the current working game, the component will fire a corresponding event to registered listeners (see requirement 1.2.6.4).

1.2.6.2 Set Current Target

The component will provide a method, intended to be exposed for scripting, by which a web page will specify an SHA-1 hash of the text of the current target identifier, in the form of a 40-character string of hexadecimal digits, and in integer sequence number. The component will persist these values locally, across browser restarts, until they are changed again via this same method.

1.2.6.3 Test Object

The component will provide a method, intended to be exposed for scripting, by which the host browser extension will test whether a particular object is the ‘current target’ (see requirement 1.2.6.2).

- The component will expect a DOM element as an argument (as provided by JavaScript / XUL / XBL).
- It will extract the text content (but not markup) of the element and all child elements, in document order, removing leading and trailing whitespace (blank, tab, CR, and LF characters), normalizing all sequences of whitespace to a single space, and folding all characters to lower case.
- It will encode the resulting text according to UTF-8, construct the SHA-1 hash, and compare it to the current target hash.
- If the hashes match, then the component will open or reuse a popup window to display the response to an HTTP request issued to a configurable URL on the Orpheus server, in much the same way described in requirement 1.2.3. The current game ID (see **Error!**

Reference source not found.), domain (as discussed in 1.2.4.1), target sequence number (1.2.6.2), and a hexadecimal digit string representing the computed hash will be specified in the request URL as query parameters of configurable name.

1.2.6.4 Event Listeners

The component will provide support for event listeners to be installed via the XUL and / or XBL documents defining the host browser extension. Listeners are likely to be implemented in JavaScript. Specific events supported are described elsewhere among these requirements, but include

- User successfully logs in
- User logs out
- Working game changed

1.2.6.5 Immediate Polling

The component will provide a method, intended to be exposed for scripting, by which a web page can instruct the component to immediately poll for messages in the same manner as the periodic polls described by requirement 1.2.5.

1.2.6.6 Test for Popup Window

The component will provide a function, intended to be accessed by scripts, that tests whether the browser window containing the current document is a popup window opened by this component (per requirement 1.2.7).

1.2.7 Pop up Windows

The component will have the ability to pop up new windows of the host Firefox instance, with specified dimensions and window decorations (buttons, toolbars, etc.), and initially either loading content from a specified URL or displaying a document provided to it. The scripting interface provided by this component will be accessible in the new window and will, directly or indirectly, be associated with the component instance that opened the window (as opposed to some new instance). The component will re-use a previously opened (by this component) popup if there is one, rather than opening a new window.

1.2.8 Browser Compatibility

The component will be compatible with Mozilla Firefox version 1.0 and above.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to enable Mozilla Firefox browsers as clients for the Orpheus application.

1.5 Future Component Direction

Additional handlers and supporting classes will be added as changes to the application require.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

None

2.1.3 Environment Requirements

- Development language: Java1.4
- Compile target: Java1.4

2.1.4 Package Structure

com.orpheus.plugin.firefox

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

URLs and for all requests to be made to the server

Request parameter names where required

Polling interval

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

Mozilla Gecko 1.7 / 1.8

3.2.2 TopCoder Software Component Dependencies:

Hashing Utility 1.0

Bloom Filter 1.1

RSS Generator 2.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None

3.2.4 QA Environment:

- Firefox 1.0.2 and 1.5.0.7
- Windows XP
- Windows 2000

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.