Game Handlers Logic 1.0 Component Specification

1. Design

The Orpheus Game Logic components provide business logic in support of game play and game data manipulation tasks performed by the Orpheus application. For the most part this involves providing `Handler` implementations conformant to the specifications of the Front Controller component version 2.1. This game logic component focuses on extracting game information from the persistence component on behalf of view generators.

This design provides nine implementations to the Handler interface from the Front Controller component according to the requirement. They all follow the handler's contract specified by the Front Controller component - concrete handler must have two constructors, one is used to create the handler programmatically, and the other is used by Front Controller to instantiate it from the xml configuration file.

All handlers in this design don't interact with the EJB interfaces directly, a GameDataHelper class is provided to hide EJB details from all handlers, and this makes the component much easier for maintenance. And basically, all handlers implement certain business logics, and they use the Game Data Persistence component to retrieve the persisted data. Refer to the class doc of each handler for more detail.

Improvements:

- From https://software.topcoder.com/forum/c_forum_message.jsp?f=24073058&r=24146441 forum post, PM confirms it is optional to support both remote and local home interfaces for Game Data Persistence component, but this design supports both home interfaces, and provides a configurable parameter in GameDataHelper to let user choose the preferred one.
- TestTargetObjectHandler, UserGamesHandler, and TestDomainHandler would return a proper result code if user is not logged in. This behavior is never mentioned in the requirement, but this feature is quite important as we normally would like to redirect user to the login page if he/she is not logged, rather than let him/her to see an error page.

1.1 Design Patterns

Singleton Pattern: The GameDataHelper class implements this pattern.

1.2 Industry Standards

Java Servlet API 2.4, XML, EJB

1.3 Required Algorithms

SPECIAL NOTES:

- As all methods' documentations contain full details about how they should be implemented, I feel it is unnecessary to duplicate the algorithms like get-leader-board in this section. Since it would confuse developers if they contain discrepancies accidentally.
- The value of handler's type attribute ('x') is configured in Front Controller component's global configuration file.

The xml Element passed in the handler's constructor must follow the DTDs defined below, but developers can choose to validate the xml or not. (You can assume the xml element is valid, and throw IAE if it is not).

Since the xml element used in this design is relatively simple, please use DOM API to extract node values.

1.3.1 Xml element for ActiveGamesHandler

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (games_key)>
```

Here are the details of the nodes in the xml element:

- The games_key node's value is used as the key for an array of Game objects to be stored in the request attributes.

Here is an example of the xml element:

```
<handler type="x">
  <games_key>games</games_key>
</handler>
```

1.3.2 Xml element for TestTargetObjectHandler

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (game_id_param_key, domain_name_param_key,
    sequence_number_param_key,
    text_param_key, test_failed_result_code, not_logged_in_result_code)>
```

Here are the details of the nodes in the xml element:

- The game_id_param_key node's value is used to get the game id value from request parameter.
- The domain_name_param_key node's value is used to get the domain name from request parameter.
- The sequence_number_param_key node's value is used to get the sequence number from request parameter.
- The text_param_key node's value is used to get the text from request parameter.
- The test_failed_validation_result_code node's value is returned in execute method if the validation fails.
- The not_logged_in_result_code node's value is returned in execute method if the user is not logged in.

Here is an example of the xml element:

```
<handler type="x">
  <game_id_param_key>gameId</game_id_param_key>
  <domain_name_param_key>domainName</domain_name_param_key>
  <sequence_number_param_key>seqNo</sequence_number_param_key>
  <text_param_key>text</text_param_key>

  <test_failed_result_code>test_failed</test_failed_result_code>
  <not_logged_in_result_code>not_logged_in</not_logged_in_result_code>
</handler>
```

1.3.3 Xml element for UserGamesHandler

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (games_key, not_logged_in_result_code)>
```

Here are the details of the nodes in the xml element:

- The games_key node's value is used as the key for an array of Game objects to be stored in the request attributes.

- The not_logged_in_result_code node's value is returned in execute method if the user is not logged in.

Here is an example of the xml element:

```
<handler type="x">
  <games_key>games</games_key>
  <not_logged_in_result_code>not_logged_in</not_logged_in_result_code>
</handler>
```

1.3.4 *Xml element for TestDomainHandler*

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (domain_name_param_key, games_key,
  not_logged_in_result_code)>
```

Here are the details of the nodes in the xml element:

- The domain_name_param_key node's value is used to get the domain name from request parameter.
- The games_key node's value is used as the key for an array of Game objects to be stored in the request attributes.
- The not_logged_in_result_code node's value is returned in execute method if the user is not logged in.

Here is an example of the xml element:

```
<handler type="x">
  <domain_name_param_key>domainName</domain_name_param_key>
  <games_key>games</games_key>
  <not_logged_in_result_code>not_logged_in</not_logged_in_result_code>
</handler>
```

1.3.5 *Xml element for UpcomingDomainsHandler*

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (game_id_param_key, domains_key)>
```

Here are the details of the nodes in the xml element:

- The game_id_param_key node's value is used to get the game id value from request parameter.
- The domains_key node's value is used as the key to store an array of Domain objects into request attributes.

Here is an example of the xml element:

```
<handler type="x">
    <game_id_param_key>gameId</game_id_param_key>
    <domains_key>domains</domains_key>
</handler>
```

1.3.6 *Xml element for UnlockedDomainsHandler*

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (game_id_param_key, domains_key)>
```

Here are the details of the nodes in the xml element:

- The game_id_param_key node's value is used to get the game id value from request parameter.
- The domains_key node's value is used as the key to store an array of Domain objects into request attributes.

Here is an example of the xml element:

```
<handler type="x">
    <game_id_param_key>gameId</game_id_param_key>
    <domains_key>domains</domains_key>
</handler>
```

1.3.7 *Xml element for GameDetailsHandler*

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (game_id_param_key, game_detail_key)>
```

Here are the details of the nodes in the xml element:

- The game_id_param_key node's value is used to get the game id value from request parameter.
- The game_detail_key node's value is used as the key to store a Game object into request attributes.

Here is an example of the xml element:

```
<handler type="x">
    <game_id_param_key>gameId</game_id_param_key>
    <game_detail_key>game</game_detail_key>
</handler>
```

1.3.8 *Xml element for SlotValidationHandler*

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (game_id_param_key, slot_id_param_key,
        validation_failed_result_code)>
```

Here are the details of the nodes in the xml element:

- The game_id_param_key node's value is used to get the game id value from request parameter.
- The slot_id_param_key node's value is used to get the slot id value from request parameter.
- The validation_failed_result_code node's value is returned in execute method if the validation fails.

Here is an example of the xml element:

```
<handler type="x">
    <game_id_param_key>gameId</game_id_param_key>
    <slot_id_param_key>slotId</slot_id_param_key>

<validation_failed_result_code>validation_failed</validation_failed_resul
t_code>
</handler>
```

1.3.9 *Xml element for LeaderBoardHandler*

The DTD of the xml element: (Only DTD for branch nodes are shown)

```
<!ELEMENT handler (game_id_param_key, profiles_key, max_leaders,
             object_factory_namespace, user_profile_manager_token)>
```

Here are the details of the nodes in the xml element:

- The game_id_param_key node's value is used to get the game id value from request parameter.
- The profiles_key node's value is used as the key to store an array of UserProfile objects into request attributes.
- The max_leaders node's value represents the maximum number of leaders should be returned on the board.
- The object_factory_namespace node's value is used to create ObjectFactory object.
- The user_profile_manager_token node's value is used to create the UserProfileManager object through ObjectFactory.

Here is an example of the xml element:

```
<handler type="x">
  <game_id_param_key>gameId</game_id_param_key>
  <profiles_key>profiles</profiles_key>
  <max_leaders>10</max_leaders>


<object_factory_namespace>com.topcoder.web.user</object_factory_namespace
>

<user_profile_manager_token>user_profile_manager</user_profile_manager_to
ken>
</handler>
```

1.4 Component Class Overview

1.4.1 Package com.orpheus.game

- **ActiveGamesHandler**: ActiveGamesHandler class implements the Handler interface from the FrontController component. It determines which are the currently active games and loads an array containing them (as Game instances) into a request attribute of configurable name.
- **TestTargetObjectHandler**: TestTargetObjectHandler class implements the Handler interface from the Front Controller component. It tests target objects to determine whether they are correct. The current game ID, domain name, target sequence number, and the UTF-8 encoded text from which the hash was computed will be read from request parameters of configurable name. The handler finds the hosting slot corresponding to the provided game ID and domain for the logged-in player via the game persistence component, and will use the provided sequence number to choose from it the domain target object to be compared with the provided text. It will return a configurable result string if the texts differ, or will return null if they match.
- **UserGamesHandler**: UserGamesHandler class implements the Handler interface from the Front Controller component. It determines which games the current (logged-in) user is registered for and loads an array containing them (as Game instances) into a request attribute of configurable name.
- **TestDomainHandler**: TestDomainsHandler class implements the Handler interface from the Front Controller component. It determines in which active games the specified domain is a current or past host for the current user, creates an array

containing the corresponding Game objects, and assigns the array to a request attribute of configurable name.

- **UpcomingDomainsHandler**: UpcomingDomainsHandler class implements the Handler interface from the Front Controller component. It will load a game id value from the request parameter, and then use it to find the domains from the current and next hosting blocks of the current game that have not yet been unlocked; it will assign an array containing their corresponding Domain objects to a request attribute of configurable name.
- **UnlockedDomainsHandler**: UnlockedDomainsHandler class implements the Handler interface from the Front Controller component. It finds the domains unlocked so far in a specified game and assigns an array containing their Domain representations to a request attribute of configurable name. The game for which domains are requested will be identified by its unique ID, parsed from a request parameter of configurable name.
- **GameDetailHandler**: GameDetailHandler class implements the Handler interface from the Front Controller component. It looks up a Game by its ID and loads it into a request attribute of configurable name. The game ID will be parsed from a request parameter of configurable name.
- **SlotValidationHandler**: SlotValidationHandler class implements the Handler interface from the Front Controller component. It verifies that a specified game is currently active, and that in it the specified slot is the current host. The game and slot IDs to validate are taken from request parameters of configurable name, and a configurable result string will be returned if this validation fails.
- **LeaderBoardHandler**: LeaderBoardHandler class implements the Handler interface from the Front Controller component. It determines the current leaders in a specified game, creates an array containing them (in order), and assigns the array to a request attribute of configurable name. The maximum number of leaders to rank will be set as a configuration parameter.
- **GameDataHelper**: GameDataHelper class encapsulates all EJB calls to the Game Data Persistence Interface, so that the handler implementations don't have to interact with the EJB interfaces directly, which makes them simpler and easier to maintain.
- **PlayerInfo**: PlayerInfo class is a private inner static class of GameDataHelper class. It encapsulates necessary attributes used to sort players on the leader board.
- **PlayerInfoComparator**: PlayerInfoComparator class implements the Comparator interface; it is a private inner static class of GameDataHelper class. This class is used to sort the PlayerInfo objects in ascending order according the rules defined in requirement spec.

1.5 Component Exception Definitions

1.5.1 Custom Exceptions

All handlers will throw the HandlerExecutionException from the Front Controller component.

- **GameDataConfigurationException**: GameDataConfigurationException is thrown from GameDataHelper's static initializer if the any property is not configured properly.

1.5.2 System Exceptions

- **IllegalArgumentException**: Used wherever empty String argument is used while not acceptable. Normally an empty String is checked with trimmed result. It also used when the argument contains invalid data.

- **ClassCastException**: Thrown from the PlayerInfoComparator if the given objects are not type of PlayerInfo.

1.6 Thread Safety

This component is thread-safe, and since the handlers can be called by multiple users from different threads, while they are deployed together with the Front Controller in a servlet container, this component MUST be thread-safe.

All handlers provided by this component are immutable, so their thread-safety depends on whether the used components are thread-safe or not.

GameDataHelper class is used by all handlers, and this class is also responsible for making EJB calls to the Game Data Persistence component, as the EJB calls are all thread-safe, and GameDataHelper itself is immutable, all the TCS components used by it are thread-safe, and its private inner static class – PlayerInfo (which is not thread-safe itself) is used in a thread-safe way by this class (the PlayerInfo object is never shared in different threads), and its another private inner static class – PlayerInfoComparator is stateless, so this class can be safely used by handlers from different threads.

The TestTargetObjectHandler, UserGamesHandler, and TestDomainHandler classes also use the LoginHandler from Web Application User Logic component to get the UserProfile from session, as LoginHandler is thread-safe, so these handlers are thread-safe too.

The LeaderBoardHandler class also uses the UserProfileManager interface to get UserProfile objects from persistence, and it's expected that we will use a thread-safe implementation of the UserProfileManager (at least its getUserProfile method should be thread-safe) to ensure it works properly in this component.

All the other handlers solely use the GameDataHelper class, so they are thread-safe.

NOTE: Although UserProfile used in this design is not thread-safe, as it is read-only in this design, so it won't cause any thread-safety problem.

2. Environment Requirements

2.1 Environment

Java 1.4, Servlet Container supports Java Servlet API 2.4, and the EJB container for the deployment of Game Data Persistence component.

2.2 TopCoder Software Components

- Configuration Manager 2.1.5: used to support component configuration.
- Front Controller 2.1: all handler implementations from this design implements the Handler interface from Front Controller.
- User Profile 1.0: UserProfile class is used to store user information.
- User Profile Manager 1.0: used to get the UserProfile from persistence.
- Object Factory 2.0.1 used in LeaderBoardHandler to create UserProfileManager object.
- Orpheus Game Persistence 1.0: all EJB and entity interfaces used in this design are from this component.
- JNDI Utility 1.0: used to get JNDI object in this design.
- Base Exception 1.0: the custom exception in this design extends the base class from this component.

- Web Application User Logic 1.0: the LoginHandler class used in this design is from this component.

NOTE: Site Validator 1.0, Messenger Framework 1.0, and Puzzle Framework 1.0 might be used in the whole application, but they are not used directly in this design.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.orpheus.game

3.2 Configuration Parameters

Used in GameDataHelper, loaded by ConfigManager, and the namespace for this parameter is: com.orpheus.game.GameDataHelper.

Parameter	Description	Values
use_remote_home	Indicate remote home interface should be used or not. If it is true, remote home interface is used; otherwise, local home interface is used.	Must be either “true” or “false”. optional, default to “true" (without quotes)
game_data_jndi_name	The jndi name to get the remote/local home interface object.	Must be non-null, non-empty string. Required.
jndi_context_name	The context name used to get specific context from the JNDIUtility component.	Must be non-null, non-empty string. Optional, default to “default”.

3.3 Dependencies Configuration

The Front Controller and Game Data Persistence components should be properly deployed.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Follow configuration instructions.

4.3 Demo

4.3.1 The global configuration file for all handlers

```
<global>
  <actions-def>
     <!-- the default action from Front Controller component -->
      <action-def name="default">
```

```
        com.topcoder.web.FrontController.DefaultAction
    </action-def>
  </actions-def>

  <handlers-def>
    <!-- define the ActiveGamesHandler -->
    <handler-def name="active_games">
       com.orpheus.game.ActiveGamesHandler
    </handler-def>
    <!-- define the TestTargetObjectHandler. -->
    <handler-def name="test_target_object">
        com.orpheus.game.TestTargetObjectHandler
    </handler-def>
    <!-- define the UserGamesHandler -->
    <handler-def name="user_games">
        com.orpheus.game.UserGamesHandler
    </handler-def>
    <!-- default the TestDomainHandler -->
    <handler-def name="test_domain">
        com.orpheus.game.TestDomainHandler
    </handler-def>
    <!-- define the UpcomingDomainsHandler -->
    <handler-def name="upcoming_domains">
        com.orpheus.game.UpcomingDomainsHandler
    </handler-def>
    <!-- define the UnlockedDomainsHandler -->
    <handler-def name="unlocked_domains">
        com.orpheus.game.UnlockedDomainsHandler
    </handler-def>
    <!-- define the GameDetailHandler -->
    <handler-def name="game_detail">
        com.orpheus.game.GameDetailHandler
    </handler-def>
    <!-- define the LeaderBoardHandler -->
    <handler-def name="leader_board">
        com.orpheus.game.LeaderBoardHandler
    </handler-def>
    <!-- define the SlotValidationHandler -->
    <handler-def name="slot_validation">
        com.orpheus.game.SlotValidationHandler
    </handler-def>
  </handlers-def>

  <results-def>
    <!-- the forward result from Front Controller component -->
    <result-def name="forward">
       com.topcoder.web.Front Controller.results.ForwardResult
    </result-def>
  </results-def>
 </global>
```

4.3.2 *Configure ActiveGamesHandler in Front Controller*

```
<action name="ActiveGames" type="default" url-pattern="/activeGames"
priority="10">
<handler type="active_games">
  <games_key>games</games_key>
</handler>

  <result name="success" type="forward">
    <forward-url>/active_games.jsp</forward-url>
  </result>
</action>
```

Assume the user sends a request to the /activeGames path, an array of Game objects is stored in request attributes, which could be rendered on the active_games.jsp page.

4.3.3 *Configure TestTargetObjectHandler in Front Controller*

```
<action name="TestTargetObject" type="default" url-
pattern="/testTargetObject" priority="10">
<handler type="test_target_object">
  <game_id_param_key>gameId</game_id_param_key>
  <domain_name_param_key>domainName</domain_name_param_key>
  <sequence_number_param_key>seqNo</sequence_number_param_key>
  <text_param_key>text</text_param_key>

  <test_failed_result_code>test_failed</test_failed_result_code>
  <not_logged_in_result_code>not_logged_in</not_logged_in_result_code>
</handler>

  <result name="not_logged_in" type="forward">
    <forward-url>/login.jsp</forward-url>
  </result>
  <result name="test_failed" type="forward">
    <forward-url>/failure.jsp</forward-url>
  </result>
  <result name="success" type="forward">
    <forward-url>/success.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /testTargetObject path with proper parameter values, if user is not logged in, the login.jsp will be shown to user to allow him/her to login first. If user has logged in, the parameter values in the request would be validated, and if the test fails, user will be forward to failure.jsp page, otherwise, user will be forward to success.jsp page.

4.3.4 *Configure UserGamesHandler in Front Controller*

```
<action name="UserGames" type="default" url-pattern="/userGames"
priority="10">
<handler type="user_games">
  <games_key>games</games_key>
  <not_logged_in_result_code>not_logged_in</not_logged_in_result_code>
</handler>

  <result name="not_logged_in" type="forward">
    <forward-url>/login.jsp</forward-url>
  </result>

  <result name="success" type="forward">
    <forward-url>/user_games.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /userGames path, if the user is not logged in, login.jsp page will be shown to him/her to login. If the user is logged in, an array of Game objects current user participates will be stored in request attributes and can be rendered on the user_games.jsp page.

4.3.5 *Configure TestDomainHandler in Front Controller*

```
<action name="TestDomain" type="default" url-pattern="/testDomain"
priority="10">
  <handler type="test_domain">
    <domain_name_param_key>domainName</domain_name_param_key>
    <games_key>games</games_key>
    <not_logged_in_result_code>not_logged_in</not_logged_in_result_code>
  </handler>

  <result name="not_logged_in" type="forward">
    <forward-url>/login.jsp</forward-url>
  </result>

  <result name="success" type="forward">
    <forward-url>/domain_games.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /testDomain path with a domain name parameter value, if the user is not logged in, login.jsp page will be shown to him/her to login. If the user is logged in, an array of active Game objects will be saved to request attributes and can be rendered on the domain_games.jsp page.

4.3.6 *Configure UpcomingDomainsHandler in Front Controller*

```
<action name="UpcomingDomains" type="default" url-
pattern="/upcomingDomains" priority="10">
  <handler type="upcoming_domains">
    <game_id_param_key>gameId</game_id_param_key>
    <domains_key>domains</domains_key>
  </handler>

  <result name="success" type="forward">
    <forward-url>/domains.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /upcomingDomains path with a gameId parameter value, an array of upcoming Domain objects will be saved to request attributes, and can be rendered on the domains.jsp page.

4.3.7 *Configure UnlockedDomainsHandler in Front Controller*

```
<action name="UnlockedDomains" type="default" url-
pattern="/ulockedDomains" priority="10">
  <handler type="unlocked_domains">
    <game_id_param_key>gameId</game_id_param_key>
    <domains_key>domains</domains_key>
  </handler>

  <result name="success" type="forward">
    <forward-url>/domains.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /unlockedDomains path with a gameId parameter value, an array of unlocked Domain objects will be saved to request attributes, and can be rendered on the domains.jsp page.

4.3.8 *Configure GameDetailHandler in Front Controller*

```
<action name="GameDetail" type="default" url-pattern="/gameDetail"
priority="10">
  <handler type="game_detail">
    <game_id_param_key>gameId</game_id_param_key>
    <game_detail_key>game</game_detail_key>
  </handler>

  <result name="success" type="forward">
    <forward-url>/game.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /gameDetail path with a gameId parameter value, corresponding Game object will be retrieved and saved to request attributes, and it can be rendered on the game.jsp page.

4.3.9 *Configure SlotValidationHandler in Front Controller*

```
<action name="SlotValidation" type="default" url-
pattern="/slotValidation" priority="10">
  <handler type="slot_validation">
    <game_id_param_key>gameId</game_id_param_key>
    <slot_id_param_key>slotId</slot_id_param_key>

  <validation_failed_result_code>validation_failed</validation_failed_res
  ult_code>
  </handler>

  <result name="validation_failed" type="forward">
    <forward-url>/failure.jsp</forward-url>
  </result>

  <result name="success" type="forward">
    <forward-url>/success.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /slotValidation path with a gameId and slotId parameter values, the corresponding slot in the corresponding game will be validated, and if the validation succeeds, Front Controller would forward to success.jsp page, otherwise, it would forward to failure.jsp page.

4.3.10 *Configure LeaderBoardHandler in Front Controller*

```
<action name="LeaderBoard" type="default" url-pattern="/leaderBoard"
priority="10">
  <handler type="slot_validation">
    <game_id_param_key>gameId</game_id_param_key>
    <profiles_key>profiles</profiles_key>
    <max_leaders>10</max_leaders>


  <object_factory_namespace>com.topcoder.web.user</object_factory_namespa
  ce>

  <user_profile_manager_token>user_profile_manager</user_profile_manager_
  token>
  </handler>
```

```
  <result name="success" type="forward">
    <forward-url>/leader_board.jsp</forward-url>
  </result>
</action>
```

Assume user sends a request to /leaderBoard path with a gameId parameter values, the top 10 (which is configurable in the xml element) players in the corresponding game (an array of UserProfile objects) will be saved to request attributes, and can be rendered on the leader_board.jsp page. .

4.3.11 *Create handlers programmatically*

Sometimes it might be useful to create handler programmatically, when we want to configure the actions in Front Controller dynamically. But we are NEVER supposed to call the execute method on our own, it is expected to be called by the Front Controller.

```
// create ActiveGamesHandler
ActiveGamesHandler handler1 = new ActiveGamesHandler(“games”);

// create TestTargetObjectHandler
Map ttoAttrs = new HashMap();
ttoAttrs.put(“gameIdParamKey”, “gameId”);
ttoAttrs.put(“domainNameParamKey”, “domainName”);
ttoAttrs.put(“sequenceNumberParamKey”, “seqNo”);
ttoAttrs.put(“textParamKey”, “text”);
ttoAttrs.put(“testFailedResultCode”, “test_failed”);
ttoAttrs.put(“notLoggedInResultCode”, “not_logged_in”);

TestTargetObjectHandler handler2 = new TestTargetObjectHandler(ttoAttrs);

// create UserGamesHandler
UserGamesHandler handler3 =
       new UserGamesHandler(“games”, “not_logged_in”);

// create TestDomainHandler
TestDomainHandler handler4 =
       new TestDomainHandler(“domainName”, “games”, “not_logged_in”);

// create UpcomingDomainsHandler
UpcomingDomainsHandler handler5 =
       new UpcomingDomainsHandler(“gameId”, “domains”);

// create UnlockedDomainsHandler
UnlockedDomainsHandler handler6 =
       new UnlockedDomainsHandler(“gameId”, “domains”);

// create GameDetailHandler
GameDetailHandler handler7 =
       new GameDetailHandler(“gameId”, “gameDetail”);

// create LeaderBoardHandler
Map lbAttrs = new HashMap();
```

```
lbAttrs.put(“gameIdParamKey”, “gameId”);
lbAttrs.put(“profilesKey”, “profiles”);
lbAttrs.put(“maxLeaders”, new Integer(10));

// profileManager object is a UserProfileManager object provided by user
lbAttrs.put(“profileManager”, profileManager);
LeaderBoardHandler handler8 = new LeaderBoardHandler(lbAttrs);

// create SlotValidationHandler
SlotValidationHandler handler9 =
      new SlotValidationHandler(“gameId”, “slotId”, “validation_failed”);
```

5 Future Enhancements

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