Game Operation Logic 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 generally on handling user actions that require updating persistent application data, or that involve non-trivial computation or data processing within handler or result objects.

1.1 Design Patterns

None.

1.2 Industry Standards

Java Servlet API 2.4

1.3 Required Algorithms

The xml Element passed to the handler's constructor must follow the DTD defined below, however developers can choose to validate the xml or not. (We can assume the xml element is valid, and throw IAE if failed to extract what we want).
The xml element structure is relatively simple, please use DOM API to extract node values.

1.3.1 RegisterGameHandler

The DTD of the xml element:

```
<!ELEMENT handler (game_id_param_key)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT game_id_param_key (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x">
   <game_id_param_key>
      game_id
   </game_id_param_key>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design.

The game_id_param_key node's value represents the http request parameter name to get the game id.

Assume user logs in, and the game id is 1, this handler will get the user id and game id from the http request parameter, and invoke the EJB object to register the user with the given game id.

1.3.2 KeySubmissionHandler

The DTD of the xml element:

```
<!ELEMENT handler (game_id_param_key,submission_param_key,
   max_failure_count,inactive_game_result, failure_count_not_met_result,
   failure_count_exceeded_result)>
<!ATTLIST handler type CDATA #REQUIRED>

<!ELEMENT game_id_param_key (#PCDATA)>
<!ELEMENT submission_param_key (#PCDATA)>
<!ELEMENT max_failure_count (#PCDATA)>
<!ELEMENT inactive_game_result (#PCDATA)>
<!ELEMENT failure_count_not_met_result (#PCDATA)>
<!ELEMENT failure_count_exceeded_result (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x">
    <game_id_param_key>
       Game_id
    </game_id_param_key>
    <submission_param_key>
       submissions
    </submission_param_key>
    <max_failure_count>
       10
    <max_failure_count>
    <inactive_game_result>
       inactive_game_result
    </inactive_game_result>
    <failure_count_not_met_result>
       count_not_met_result
    </failure_count_not_met_result>
    <failure_count_exceeded_result>
       count_exceeded_result
    </failure_count_exceeded_result>
</handler>
```

Following is simple explanation of the above XML structure.

The handler’s type attribute is required by Front Controller component, it won’t be used in this design.

The game_id_param_key node’s value represents the http request parameter name to get the game id

The submission_param_key node value represents the http request parameter name to get the key submissions

The max_failure_count node represents the max failure count of the comparision of the submitted submissions with these retrieved from EJB

The failure_count_not_met_result node represents the result name this handler will return if the max count is not met

The failure_count_exceeded_result node represents the result name this handler will return if the max count is met or exceeded.

This handler processes key submissions. It will accept a game ID via a request parameter of configurable name, and several key strings as multiple values of another request parameter, also of configurable name. It will determine whether the submitted keys match those recorded for the currently logged-in player for the specified game for the most recently completed (by any player) *k* hosting slots, where *k* is the key count of the specified game.

The handler will maintain a count, in the player's session, of the number of failures to submit matching keys for the specified game since the last time its hosting slot changed. (The ID of the most recently-completed hosting slot in that game is one possible measure of whether the hosting slot has changed.)

Contingencies and results:

- If the specified game is not active then the handler returns a configurable result code characteristic of the problem, regardless of the values of the keys
- If the keys do not match then the player's failure count for the game and slot is incremented. The handler then returns a configurable result string characteristic of whether or not the failure count meets or exceeds a configurable threshold value.
- If the keys do match then the handler records a game completion for the player in the specified game (via the game data persistence component), then returns null

1.3.3 PuzzleSolutionHandler

The DTD of the xml element:

```
<!ELEMENT handler (puzzle_id_request_attribute_key,
   media_type_request_attribute_key,
   puzzle_string_request_attribute_key,
   solutiontester_base_name,
   incorrect_solution_result)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT puzzle_id_request_attribute_key (#PCDATA)>
<!ELEMENT media_type_request_attribute_key (#PCDATA)>
<!ELEMENT puzzle_string_request_attribute_key (#PCDATA)>
<!ELEMENT solutiontester_base_name (#PCDATA)>
<!ELEMENT incorrect_solution_result (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x">
  <puzzle_id_request_attribute_key>
     puzzle_id
  </puzzle_id_request_attribute_key>
  <media_type_request_attribute_key>
     media_type
  </media_type_request_attribute_key>
  <puzzle_string_request_attribute_key>
     puzzle_string
  </puzzle_string_request_attribute_key>
  <solutiontester_base_name>
```

```
     base_name
  </solutiontester_base_name>
  <incorrect_solution_result>
     incorrect_solution_result
  </incorrect_solution_result>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design.

The puzzle_id_request_attribute_key node's value represents the http request attribute name to get the puzzle id

The media_type_request_attribute_key node represents the http request attribute name to get the media type.

The puzzle_string_request_attribute_key node represents the http request attribute name to store the rendered puzzle string.

The solutiontester_base_name node represents the SolutionTester base name

The incorrect_solution_result node represents the result name to return if the puzzle solution is incorrect.

This handler determines the IDs of the game and slot to which the puzzle applies from request parameters of configurable name. It then finds the appropriate solution tester:

- it looks up the ID of the slot's puzzle among the slot information,
- appends that puzzle ID to a configured prefix to obtain the name of the session attribute containing the appropriate `SolutionTester`, and
- retrieves the tester from the session via the computed attribute name

The handler will pass the request's full parameter map to the tester. If the solution is correct then the handler records a game completion for the currently logged-in player in the game whose ID is available from a request parameter of configurable name, and the handler returns null.

If the solution is incorrect then the game is forced to advance to the next host, and a slot completion is recorded for the current player in the specified slot, the `SlotCompletion` object is assigned to a request attribute of configurable name, and the handler returns a configurable result code string.

1.3.4 PuzzleRenderingHandler

The DTD of the xml element:

```
<!ELEMENT handler (game_id_param_key,slot_id_param_key,
   solutiontester_base_name)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT game_id_param_key (#PCDATA)>
<!ELEMENT slot_id_param_key (#PCDATA)>
<!ELEMENT solutiontester_base_name (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x">
    <game_id_param_key>
```

```
        puzzle_id
    </game_id_param_key>
    <slot_id_param_key>
        Slot_id
    </slot_id_param_key>
    <solutiontester_base_name>
        base_name
    </solutiontester_base_name>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design.

The puzzle_id_param_key node's value represents the http request parameter name to get the puzzle id

The slot_id_param_key node represents the http request parameter name to get the slot id

The solutiontester_base_name node represents the SolutionTester base name.

This handler produces a rendition of a puzzle in `String` form and attaches it to the request as an attribute of configurable name. The handler will be configured also with the name of a request attribute from which to obtain the ID of the puzzle to render (as a `Long`), the media type in which to render it (a `String`), the name of the application context attribute from which to obtain a `PuzzleTypeSource`, and the base name of a session attribute to which to assign a `SolutionTester`. The handler operates in this fashion:

- It obtains from game data persistence a `PuzzleData` for the specified ID.
- It extracts the puzzle type name from among the `PuzzleData`'s attributes, and feeds it to a `PuzzleTypeSource` to obtain the appropriate `PuzzleType`.
- It obtains a renderer for the specified medium from the PuzzleType, and uses it to render the puzzle.
- It assigns the `SolutionTester` resulting from the rendering to a session attribute named by concatenating the configured base attribute name with the puzzle ID.
- It assigns the rendered puzzle, as a String, as a request attribute with the configured name.

1.3.5 *MessagePollResult*

The DTD of the xml element:

```
<!ELEMENT result (date_param_key, catetory_names)>
<!ATTLIST result type CDATA #REQUIRED>
<!ELEMENT game_id_param_key (#PCDATA)>
<!ELEMENT catetory_names (value+)>
<!ELEMENT value (#PCDATA)>
```

Follow is a sample xml:

```
<result name="x">
  <date_param_key>
```

```
     date
  </date_param_key>
  <catetory_names>
    <value>some category name</value>
    <value>some category name </value>
    <value>some category name </value>
  </category_names>
</result>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design.

The date node's value represents the http request parameter name to get the date string

The category_names node contains an array of category names that will be used to construct the SearchCriteria

This result will obtain a `DataStore` from an application context attribute of configurable name and use it to construct an Atom 1.0 feed document, which it returns (in XML format) as the HTTP response entity. It will select items for the feed as follows:

- The `Result` will be configured with a request parameter name with which it will accept an item update date and time. If the request specifies a parameter of that name then this `Result` will parse it as a `java.util.Date` in ISO 8601:2004 format (http://en.wikipedia.org/wiki/ISO_8601), and only items with a creation or update date strictly later than the parsed date will be included in the resulting feed.
- The `Result` will determine the games for which the requesting player (the one currently logged-in) is registered; the name of each will be used as an item category, and items in any of those categories will be included in the feed, subject to the date restriction already described.
- The `Result` will be configured with zero or more category names for categories whose items are always included in the feed, subject to the date restriction criterion.

The response will be assigned the content type registered for Atom, "application/atom+xml"

1.3.6 WinnerDataHandler

The DTD of the xml element:

```
<!ELEMENT handler (object_factory,profile_type_names,
   profile_property_param_names, profile_property_names)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT object_factory (namespace, user_profile_manager)>
<!ELEMENT namespace (#PCDATA)>
<!ELEMENT user_profile_manager (#PCDATA)>
<!ELEMENT profile_type_names (value+)>
<!ELEMENT profile_property_param_names (value+)>
<!ELEMENT profile_property_names (value+)>
<!ELEMENT value (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x" >
```

```
    <object_factory>
      <namespace> com.orpheus.game </namespace>
      <user_profile_manager_key>
         user_profile_manager
      </user_profile_manager>
    </object_factory>
    <profile_type_names>
        <value>some value</value>
        <value> some value</value>
    </profile_type_names>
    <profile_property_param_names>
        <value>paramName1</value>
        <value>paramName2</value>
        .....
    </profile_property_param_names>
    <profile_property_names>
        <value>propertyName1</value>
        <value>propertyName2</value>
        .....
    </profile_property_names>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design.

The object_factory node contains the values to create the UserProfileManager from ObjectFactory

The profile_type_names node contains the profile type names that will be added to the UserProfile

The profile_property_param_names node contains the http request parameter names used to get the property value.

The profile_property_names node contains the profile property names.

Please note that the profile_property_param_names must have the same number of children value nodes as the profile_property_names.

This handler records contact information collected from players when they win the game. Information will be stored in the winner's User Profile, and recorded using the User Profile Manager. The handler's configuration will specify profile types that the handler must ensure are present in the profile, and will specify a map from parameter names to corresponding user profile property names. The Handler will add the profile type(s) to the UserProfile, then copy the parameter values to the profile and store it.

1.3.7 *MessageHandler*

The DTD of the xml element:

```
<!ELEMENT handler (plugin_name,attribute_names,message_property_names)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT plugin_name (#PCDATA)>
<!ELEMENT attribute_names (value+)>
<!ATTLIST attribute_names scope CDATA #REQUIRED>
<!ELEMENT message_property_names (value+)>
<!ELEMENT value (#PCDATA)>
```

```
<!ATTLIST value scope CDATA #REQUIRED>
```

Follow is a sample xml:

```
<handler type="x" >
    <plugin_name>some name</plugin_name>
    <attribute_names>
        <value scope=”request_parameter”>paramName1</value>
        <value scope=”application”>attributeName2</value>
        .....
    </attribute_names>
    <message_property_names>
        <value>some value</value>
        <value>some value</value>
        .........
    </message_property_names>
</handler>
```

Following is simple explanation of the above XML structure.

The handler’s type attribute is required by Front Controller component, it won’t be used in this design.

The plugin_name node’s value represents the message plugin name used to get the MessagePlugin

The attribute_names node’s values are used to get the corresponding message property value from the request parameter or attribute, session attribute, application attribute, or ActionContext attribute, dependent on the scope attribute. The scope value must be one of the following values:

request_parameter, request, session, application, action_context.

The message_property_names’ values service as the message property name to keep the property value.

Please note that the attribute_names must have the same number of children value nodes as the message_property_names.

This handler supports general-purpose messaging between the application and users via the Messenger Framework. It is configured with the name of a MessengerPlugin and with one or more mappings between message property names and request parameter names or request / session / application / action context attribute names, dependent on the scope value. The handler obtains a MessengerPlugin instance, creates a MessageAPI instance, assigns values to the message parameters as directed by its configuration, then dispatches the message via the MessengerPlugin.

1.3.8 PluginDownloadHandler

The DTD of the xml element:

```
<!ELEMENT handler (plugin_name_param_key)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT plugin_name_param_key (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x">
   < plugin_name_param_key >
   some key
```

```
    </ plugin_name_param_key >
</handler>
```

Following is simple explanation of the above XML structure.

The handler’s type attribute is required by Front Controller component, it won’t be used in this design.

The plugin_name_param_key node’s value represents the http request parameter name to get the message plugin name

This handler records an instance of a plug-in download. It will use a plug-in name taken from a request parameter of configurable name to identify the plug-in downloaded to the game data persistence component.

1.4 Component Class Overview

RegisterGameHandler:

Registers the current (logged-in) user for a specified game. The game to register for will by identified by its unique ID, parsed from a request parameter of configurable name.

This class is thread safe since it does not contain any mutable state.

KeySubmissionHandler:

A Handler that processes key submissions. It will accept a game ID via a request parameter of configurable name, and several key strings as multiple values of another request parameter, also of configurable name. It will determine whether the submitted keys match those recorded for the currently logged-in player for the specified game for the most recently completed (by any player) *k* hosting slots, where *k* is the key count of the specified game.

The handler will maintain a count, in the player’s session, of the number of failures to submit matching keys for the specified game since the last time its hosting slot changed. (The ID of the most recently-completed hosting slot in that game is one possible measure of whether the hosting slot has changed.)

Contingencies and results:

- If the specified game is not active then the handler returns a configurable result code characteristic of the problem, regardless of the values of the keys
- If the keys do not match then the player’s failure count for the game and slot is incremented. The handler then returns a configurable result string characteristic of whether or not the failure count meets or exceeds a configurable threshold value.
- If the keys do match then the handler records a game completion for the player in the specified game (via the game data persistence component), then returns null

This class is thread safe since it does not contain any mutable state.

MessagePollResult

A Result for responding to message polling requests. It will obtain a `DataStore` from an application context attribute of configurable name and use it to construct an Atom 1.0 feed document, which it returns (in XML format) as the HTTP response entity. It will select items for the feed as follows:

- The `Result` will be configured with a request parameter name with which it will accept an item update date and time. If the request specifies a parameter of that name then this `Result` will parse it as a `java.util.Date` in ISO 8601:2004 format (http://en.wikipedia.org/wiki/ISO_8601), and only items with a creation or update date strictly later than the parsed date will be included in the resulting feed.
- The `Result` will determine the games for which the requesting player (the one currently logged-in) is registered; the name of each will be used as an item category, and items in any of those categories will be included in the feed, subject to the date restriction already described.
- The `Result` will be configured with zero or more category names for categories whose items are always included in the feed, subject to the date restriction criterion.

The response will be assigned the content type registered for Atom, “application/atom+xml”.

This class is thread safe since it does not contain any mutable state.

MessageHandler:

A handler that supports general-purpose messaging between the application and users via the Messenger Framework. It is configured with the name of a `MessengerPlugin` and with one or more mappings between message property names and request parameter names or request / session / application / action context attribute names. The handler obtains a `MessengerPlugin` instance, creates a `MessageAPI` instance, assigns values to the message parameters as directed by its configuration, then dispatches the message via the `MessengerPlugin`.

This class is thread safe since it does not contain any mutable state.

PuzzleRenderingHandler:

A Handler that produces a rendition of a puzzle in `String` form and attaches it to the request as an attribute of configurable name. The handler will be configured also with the name of a request attribute from which to obtain the ID of the puzzle to render (as a `Long`), the media type in which to render it (a `String`), the name of the application context attribute from which to obtain a `PuzzleTypeSource`, and the base name of a session attribute to which to assign a `SolutionTester`. The handler operates in this fashion:

- It obtains from game data persistence a `PuzzleData` for the specified ID.
- It extracts the puzzle type name from among the `PuzzleData`’s attributes, and feeds it to a `PuzzleTypeSource` to obtain the appropriate `PuzzleType`.
- It obtains a renderer for the specified medium from the PuzzleType, and uses it to render the puzzle.
- It assigns the `SolutionTester` resulting from the rendering to a session attribute named by concatenating the configured base attribute name with the

puzzle ID.

- It assigns the rendered puzzle, as a String, as a request attribute with the configured name.

This class is thread safe since it does not contain any mutable state.

PuzzleSolutionHandler:

The component will provide a `Handler` that tests whether the HTTP request parameters describe a solution to a previously presented puzzle. It determines the IDs of the game and slot to which the puzzle applies from request parameters of configurable name. It then finds the appropriate solution tester:

- it looks up the ID of the slot's puzzle among the slot information,
- appends that puzzle ID to a configured prefix to obtain the name of the session attribute containing the appropriate `SolutionTester`, and
- retrieves the tester from the session via the computed attribute name

The handler will pass the request's full parameter map to the tester. If the solution is correct then the handler records a game completion for the currently logged-in player in the game whose ID is available from a request parameter of configurable name, and the handler returns null.

If the solution is incorrect then the game is forced to advance to the next host, and a slot completion is recorded for the current player in the specified slot, the `SlotCompletion` object is assigned to a request attribute of configurable name, and the handler returns a configurable result code string.

This class is thread safe since it does not contain any mutable state.

WinnerDataHandler:

A Handler that records contact information collected from players when they win the game. Information will be stored in the winner's User Profile, and recorded using the User Profile Manager. The handler's configuration will specify profile types that the handler must ensure are present in the profile, and will specify a map from parameter names to corresponding user profile property names. The Handler will install the profile type(s) if necessary, then copy the parameter values to the profile and store it.

This class is thread safe since it does not contain any mutable state.

PluginDownloadHandler:

A Handler that records an instance of a plug-in download. It will use a plug-in name taken from a request parameter of configurable name to identify the plug-in downloaded to the game data persistence component.

This class is thread safe since it does not contain any mutable state.

GameOperationLogicUtility:

A utility class whose methods will be used by all the handlers and results in this component.

The variable values are loaded from ConfigurationManager.

This class is thread safe since it's immutable.

1.5 Component Exception Definitions

IllegalArgumentException

This exception is thrown in various methods if null object is not allowed, or the given string argument is empty. Refer to the documentation in Poseidon for more details.

NOTE: Empty string means string of zero length or string full of whitespaces.

HandlerExecutionException

It is thrown from Handler implementation classes.

ResultExecutionException

It is thrown from Result implementation classes.

1.6 Thread Safety

This component is required to be completely thread safe as the handlers in this component will be invoked by users in multi-threads. All the handler implementations do not contain any mutable state and the calls to other components are thread safe

1. The EJB calls are thread safe
2. The call to MesseganerFramework is thread-safe
3. The call to UserProfileManager is thread-safe
4. The calls to other components(like RSS Generator) will not modify any inner state, and they are all thread-safe,

Thus, all the handlers are thread-safe. The GameOperationLogicUtility and AttributeScope is immutable and also thread safe.

2. Environment Requirements

2.1 Environment

Java 1.4 or higher.

2.2 TopCoder Software Components

Configuration Manager 2.1.5

It is used to load configuration values.

Front Controller 2.1

The Handler interface comes from this component

Auction Framework 1.0

The Auction entities come from this component.

User Profile Manager 1.0

The UserProfileManager interface comes from this component.

Messenger Framework 1.0

The GameDataManager class comes from this component.

Web Application User Logic 1.0

The LoginHandler class comes from this component.

User Profile 1.0

The UserProfile class comes from this component.

JNDI Utility 1.0

The JNDIUtils class comes from this component.

RSS Generator 2.0

MessagePol Result will construct RSS Feed and store it as HTTP response.

Orpheus Game Persistence 1.0

The Game interface comes from this component.

Game Interface Logic 1.0

The GameDataManager interface comes from this component.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.orpheus.game

3.2 Configuration Parameters

Configuration values for com.orpheus.game.GameOperationLogicUtility

Parameter	**Description**	**Values**
data_store	The value is used get the DataStore instance from the ServletContext. **Required**.	Data_store
game_data_manager	The value is used get the GameDataManager instance from the ServletContext. **Required**.	GameDataManager
puzzle_type_source	The value is used get the PuzzelTypeSource instance from the ServletContext. **Required**.	PuzzleTypeSource
context_name	The initial context name	default

	used by JNDIUtils to search for the ejb home instance. **Required**.	
game_data_local	The GameData EJB local home name used by JNDIUtils to search for the ejb local home instance**.** **Required**.	local
game_data_remote	The GameData EJB remote home name used by JNDIUtils to search for the ejb remote home instance. **Required**.	remote
use_local_interface	The value indicates if the local or remote EJB interface is used. **Required**.	true/false

3.3 Dependencies Configuration

ConfigurationManager should be properly configured to make this component work.

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

Preload the configuration file into Configuration Manager. Follow demo.

4.3 Demo

```
Assume the configuration file in 3.2 is used here. And follow the
algorithm section about how to configure handler and result.
```

4.3.1 Global file for all the handlers

```
<global>
  <actions-def>
     <!-- the default action from Front Controller component -->
    <action-def name="default">
       com.topcoder.web.Front Controller.DefaultAction
     </action-def>
  </actions-def>

  <handlers-def>
   <handler-def name="RegisterGameHandler">
      com.orpheus.game.RegisterGameHandler
    </handler-def>
    <handler-def name="KeySubmissionHandler">
       com.orpheus.game.KeySubmissionHandler
    </handler-def>
    <handler-def name="PuzzleSolutionHandler">
       com.orpheus.game.PuzzleSolutionHandler
    </handler-def>
    <handler-def name="PuzzleRenderingHandler">
       com.orpheus.game.PuzzleRenderingHandler
    </handler-def>
    <handler-def name="WinnerDataHandler">
       com.orpheus.game.WinnerDataHandler
    </handler-def>
    <handler-def name="PluginDownloadHandler">
       com.orpheus.game.
    </handler-def>
    <handler-def name="MessageHandler">
       com.orpheus.game.MessageHandler
    </handler-def>

  </handlers-def>

  <global-results>
      <result name="messagePollResult">
         <date_param_key>
             date
         </date_param_key>
        <catetory_names>
            <value>some value</value>
            <value>some value</value>
            <value>some value</value>
        </category_names>
   </result>
 </global>
```

4.3.2 How to configure the specified handlers

4.3.2.1 How to configure RegisterGameHandler

```
<action name="GegGame" type="default" url-pattern="/regGame"
priority="10">
<handler type="x">
   <game_id_param_key>
      game_id
   </game_id_param_key>
</handler>
</action>
```

Assume user logs in, and the game id is 1, this handler will get the user id and game id from the http request parameter, and invoke the EJB object to register the user with the given game id.

4.3.2.2 How to configure KeySubmissionHandler

```
<action name="KeySubmission" type="default" url-pattern="/KeySubmission"
   priority="10">
<handler type="KeySubmissionHandler">
    <game_id_param_key>
       Game_id
    </game_id_param_key>
    <submission_param_key>
        submissions
    </submission_param_key>
    <max_failure_count>
        10
    <max_failure_count>
    <inactive_game_result>
        inactive_game_result
    </inactive_game_result>
    <failure_count_not_met_result>
        count_not_met_result
    </failure_count_not_met_result>
    <failure_count_exceeded_result>
        count_exceeded_result
    </failure_count_exceeded_result>
</handler>

  <result name="inactive_game_result" type="forward">
    <forward-url>/inactive_game_result.jsp</forward-url>
  </result>
  <result name="failure_count_not_met_result" type="forward">
    <forward-url>/failure_count_not_met_result.jsp</forward-url>
  </result>
  <result name="failure_count_exceeded_result" type="forward">
    <forward-url>/failure_count_exceeded_result.jsp</forward-url>
  </result>
</action>
```

This handler processes key submissions. It will accept a game ID via a request parameter of configurable name, and several key strings as multiple values of another request parameter, also of configurable name. It will determine whether the submitted keys match those recorded for the currently logged-in player for the specified game for the most recently completed (by any player) *k* hosting slots, where *k* is the key count of the specified game.

The handler will maintain a count, in the player's session, of the number of failures to submit matching keys for the specified game since the last time its hosting slot changed. (The ID of the most recently-completed hosting slot in that game is one possible measure of whether the hosting slot has changed.)

Contingencies and results:

- If the specified game is not active then the handler returns a configurable result code characteristic of the problem, regardless of the values of the keys. `According to the above configuration file, inactive_game_result is called to forward to inactive_game_result.jsp`
- If the keys do not match then the player's failure count for the game and slot is incremented. The handler then returns a configurable result string characteristic of whether or not the failure count meets or exceeds a configurable threshold value. `According to the above configuration file, if the count is exceeded, failure_count_exceeded_result_result is called to forward to failure_count_exceeded_result.jsp;if not met, failure_count_not_met result is called to forward to failue_count_not_met.jsp`
- If the keys do match then the handler records a game completion for the player in

the specified game (via the game data persistence component), then returns null

4.3.2.3 How to configure PuzzleSolutionHandler

```
<action name=" PuzzleSolutionHandler" type="default"
    url-pattern="/PuzzleSolution" priority="10">
<handler type="PuzzleSolutionHandler">
  <puzzle_id_request_attribute_key>
     puzzle_id
  </puzzle_id_request_attribute_key>
  <media_type_request_attribute_key>
     media_type
  </media_type_request_attribute_key>
  <puzzle_string_request_attribute_key>
     puzzle_string
  </puzzle_string_request_attribute_key>
  <solutiontester_base_name>
     base_name
  </solutiontester_base_name>
  <incorrect_solution_result>
     incorrect_solution_result
  </incorrect_solution_result>
</handler>

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

This handler determines the IDs of the game and slot to which the puzzle applies from request parameters of configurable name. It then finds the appropriate solution tester:

- it looks up the ID of the slot's puzzle among the slot information,
- appends that puzzle ID to a configured prefix to obtain the name of the session attribute containing the appropriate `SolutionTester`, and
- retrieves the tester from the session via the computed attribute name

The handler will pass the request's full parameter map to the tester. If the solution is correct then the handler records a game completion for the currently logged-in player in the game whose ID is available from a request parameter of configurable name, and the handler returns null.

If the solution is incorrect then the game is forced to advance to the next host, and a slot completion is recorded for the current player in the specified slot, the `SlotCompletion` object is assigned to a request attribute of configurable name, and the handler returns a configurable result code string. `According to the above configuration file, incorrect_solution_result is called to forward to incorrect_solution_result.jsp`

4.3.2.4 How to configure PuzzleRenderingHandler

```
<action     name="PuzzleRendering"     type="default"     url-pattern="/
   PuzzleRendering "priority="10">
<handler type="PuzzleRenderingHandler">
```

```
    <game_id_param_key>
         puzzle_id
    </game_id_param_key>
    <slot_id_param_key>
         Slot_id
    </slot_id_param_key>
    <solutiontester_base_name>
         base_name
    </solutiontester_base_name>
</handler>
</action>
```

This handler produces a rendition of a puzzle in `String` form and attaches it to the request as an attribute of configurable name. The handler will be configured also with the name of a request attribute from which to obtain the ID of the puzzle to render (as a `Long`), the media type in which to render it (a `String`), the name of the application context attribute from which to obtain a `PuzzleTypeSource`, and the base name of a session attribute to which to assign a `SolutionTester`. The handler operates in this fashion:

- It obtains from game data persistence a `PuzzleData` for the specified ID.
- It extracts the puzzle type name from among the `PuzzleData`'s attributes, and feeds it to a `PuzzleTypeSource` to obtain the appropriate `PuzzleType`.
- It obtains a renderer for the specified medium from the PuzzleType, and uses it to render the puzzle.
- It assigns the `SolutionTester` resulting from the rendering to a session attribute named by concatenating the configured base attribute name with the puzzle ID.
- It assigns the rendered puzzle, as a String, as a request attribute with the configured name.

4.3.2.5 How to configure MessagePollResult

```
<result name="MessagePollResult">
  <date_param_key>
     date
  </date_param_key>
  <catetory_names>
    <value>some category name</value>
    <value>some category name </value>
    <value>some category name </value>
  </category_names>
</result>
```

This result will obtain a `DataStore` from an application context attribute of configurable name and use it to construct an Atom 1.0 feed document, which it returns (in XML format) as the HTTP response entity. It will select items for the feed as follows:

- The `Result` will be configured with a request parameter name with which it will accept an item update date and time. If the request specifies a parameter of that name then this `Result` will parse it as a `java.util.Date` in ISO 8601:2004 format (http://en.wikipedia.org/wiki/ISO_8601), and only items with a creation or update date strictly later than the parsed date will be included in the resulting feed.

- The `Result` will determine the games for which the requesting player (the one currently logged-in) is registered; the name of each will be used as an item category, and items in any of those categories will be included in the feed, subject to the date restriction already described.
- The `Result` will be configured with zero or more category names for categories whose items are always included in the feed, subject to the date restriction criterion.

The response will be assigned the content type registered for Atom, “application/atom+xml”

4.3.2.6 How to configure WinnerDataHandler

```
<action    name="WinnerData"    type="default"    url-pattern="/WinnerData
   "priority="10">
<handler type="WinnerDataHandler" >
    <object_factory>
      <namespace> com.orpheus.game </namespace>
      <user_profile_manager_key>
         user_profile_manager
      </user_profile_manager>
    </object_factory>
    <profile_type_names>
        <value>some value</value>
        <value some value</value>
    </profile_type_names>
    <profile_property_param_names>
        <value>paramName1</value>
        <value>paramName2</value>
        .....
    </profile_property_param_names>
    <profile_property_names>
        <value>propertyName1</value>
        <value>propertyName2</value>
        .....
    </profile_property_names>
</handler>
</action>
```

This handler records contact information collected from players when they win the game. Information will be stored in the winner’s User Profile, and recorded using the User Profile Manager. The handler’s configuration will specify profile types that the handler must ensure are present in the profile, and will specify a map from parameter names to corresponding user profile property names. The Handler will add the profile type(s) to the UserProfile, then copy the parameter values to the profile and store it.

4.3.2.7 How to configure MessageHandler

```
<action      name="Message"      type="default"      url-pattern="/Message
   "priority="10">
<handler type="MessageHandler" >
    <plugin_name>some name</plugin_name>
    <attribute_names>
        <value scope=”request_parameter”>paramName1</value>
        <value scope=”application”>attributeName2</value>
        .....
    </attribute_names>
    <message_property_names>
        <value>some value</value>
        <value>some value</value>
        .........
    </message_property_names>
</handler>
</action>
```

This handler supports general-purpose messaging between the application and users via the Messenger Framework. It is configured with the name of a MessengerPlugin and with one or more mappings between message property names and request parameter names or request / session / application / action context attribute names, dependent on the scope value. The handler obtains a MessengerPlugin instance, creates a MessageAPI instance, assigns values to the message parameters as directed by its configuration, then dispatches the message via the MessengerPlugin.

4.3.2.8 How to configure PluginDownloadHandler

```
<action  name="PluginDownloadHanlder  "  type="default"  url-pattern="/
   PluginDownloadHanlder "priority="10">
<handler type="PluginDownloadHanlder">
   < plugin_name_param_key >
   some key
   </ plugin_name_param_key >
</handler>
</action>
```

This handler records an instance of a plug-in download. It will use a plug-in name taken from a request parameter of configurable name to identify the plug-in downloaded to the game data persistence component.

5. Future Enhancements

More handler implementation could be added.