Administration Logic 1.0 Component Specification

1. Design

The Administration Logic component provides business logic in support of user manipulation tasks performed by the Orpheus application.

This component provides support as implementations of the Handler interface defined in the Front Controller component. As such, this component will be used as a plug-in to the Front Controller component in a web application.

The component uses AdminData and GameData EJB instances to perform majority of the persistence related operations. Service Locator pattern has been used to locate these services. Service Locator implementation does not cache the home references in this design because there is a risk of the cached home reference being invalid due to application server restart or shutdown.

Almost all of the handler implementations do not throw HandlerExecutionException in case of failure situations. Instead, a mechanism of HandlerResult is used. An instance of HandlerResult with different ResultCodes is set as request attribute so that user programmer can generate different error messages and handle errors differently.

1.1 Design Patterns

Strategy pattern is indirectly supported by this component in the form of Handler implementations. One can say that strategy pattern is used in allowing a configurable SiteStatistics instance to be used.

Type safe enum pattern is used in the ResultCode and PuzzleTypeEnum classes.

Service Locator pattern is used to lookup EJB services.

Template pattern has been used in SponsorApprovalRejectionHandler, RegenerateBrainteaserOrPuzzleHandler, SponsorImageApprovalRejectionHandler, PendingWinnerApprovalRejectionHandler, DomainApprovalRejectionHandler and their sub classes.

1.2 Industry Standards

XML, Java Servlet API 2.4, EJB

1.3 Required Algorithms

Common Operations:

Many of the algorithms described in sections below require fetching of request parameters and parsing the request parameter value into a long value.

1) In case the request parameter is missing or has blank value, it is a failure condition and the handler should set a HandlerResult as a request attribute and return value of failResult from the execute() method. The Handler Result should be constructed as:
new HandlerResult(ResultCode.MISSING_PARAMETERS, “parameter xxx not given”)

2) Some algorithms require the request parameter value to be parsed into a long value. In case a NumberFormatException is thrown when parsing the value into long, it is a failure condition again, and the handler should set HandlerResult as a request attribute and return value of failResult from the execute() method. The Handler Result should be constructed as:
new HandlerResult(ResultCode.PARAMETER_NOT_LONG, “Unable to parse parameter xxx into long”)

   Similarly if request parameter cannot be parsed into integer value or date value, use

ResultCode.PARAMETER_NOT_INTEGER and PARAMETER_NOT_DATE respectively.

3) Wherever HandlerResult is asked to be instantiated and set as request attribute, the execute() method must return the value of failResult. The Handler Result is to be set as request attribute as follows:
actionContext.getRequest().setAttribute(failRequestAttrName, handlerResult)

4) In any of the Handler#execute() methods, if an exception occurs such as ServiceLocatorException or RemoteException or other exceptions thrown by other Topcoder components, create a HandlerResult instance as :
new HandlerResult(ResultCode.EXCEPTION_OCCURRED, some msg, exception)
and set the instance as a request attribute.

5) All the Handlers return null from execute() method in case of successful execution.

6) Many algorithms require the use of AdminData EJB service. To get an instance of this service, do as follows:

   a) ServiceLocator sl = new ServiceLocator();

   b) AdminDataHome home = sl.getRemoteHome(adminDataJndiName, AdminDataHome.class);
   adminDataJndiName will be available as an instance variable in the classes.

   c) AdminData adminData = home.create();

7) Similarly, many algorithms also require use of GameData EJB service. To get an instance of this service, do as follows:

   a) ServiceLocator sl = new ServiceLocator();

   b) GameDataHome home = sl.getRemoteHome(gameDataJndiName, GameDataHome.class);
   gameDataJndiName will be available as instance variable in the classes.

   c) GameData gameData = home.create();

1.3.1 AdminSummaryHandler#execute

1) Create the AdminData EJB instance [adminData] as described in Common Operations section.

2) Get the summary information :
AdminSummary summary = adminData.getAdminSummary();

3) Set the summary as request attribute
actionContext.getRequest().setAttribute(summaryRequestAttrName, summary)

4) Return null in case of successful execution.

1.3.2 PendingSponsorHandler#execute

1) Search for pending sponsors

   a) Declare a local variable : Map searchParams = new HashMap()

   b) searchParams.put(“IS_APPROVED”, null)

   c) Search for the pending sponsors using
   UserProfile[] sponsors = userProfileManager.searchUserProfiles(searchParams);
   Synchronize over userProfileManager during the method call.

   d) If sponsors.length is zero, create a SponsorDomain[] of zero length and set it as request attribute as specified in step 5. Else proceed with step 2.

2) Create the GameData EJB instance [gameData] as described in Common Operations section.

3) Create a var :
SponsorDomain[] sponsorDomains = new SponsorDomain[sponsors.length]

4) For every sponsor in [sponsors]:

   a) Domain[] domains =
   gameData.findDomainsForSponsor(((Long)sponsor.getIdentifier()).longValue());

   b) sponsorDomains[i] = new SponsorDomain(sponsors[i], domains)

5) Set the sponsor and domains as request attribute
actionContext.getRequest().setAttribute(sponsorDomainRequestAttrName, sponsorDomains)

6) return null in case of successful execution.

1.3.3 PendingSponsorDomainHandler#execute

1) Create the GameData EJB instance [gameData] as described in Common Operations section.

2) Get the domain:

   a) Get requested domain id from request parameter
   String domainId =
   actionContext.getRequest().getParameter(domainIdRequestParamName)

   b) Parse the domainId into a long value.

   c) Domain domain = gameData.getDomain(domainId)

3) Search for domain's associated sponsor
UserProfile sponsor = userProfileManager.getUserProfile(domain.getSponsorId());
Synchronize over userProfileManager during the method call.

4) Set the domain and sponsor as request attributes
request.setAttribute(domainRequestAttrName, domain)
request.setAttribute(sponsorRequestAttrName, sponsor)

5) Return null in case of successful execution.

1.3.4 SponsorApprovalRejectionHandler#execute

1) Search for sponsor

   a) Get sponsor id from request parameter
   String sponsorId =
   actionContext.getRequest().getParameter(sponsorIdRequestParamName)

   b) Parse sponsorId into a long value.

   c) UserProfile sponsor = userProfileManager.getUserProfile(sponsorId);
   Synchronize over userProfileManager during the method call.

2) If userProfile.getProperty("IS_APPROVED") is not null or blank, create a HandlerResult instance as :
new HandlerResult(ResultCode.APPROVAL_NOT_PENDING, some msg) and set the instance as a request attribute.

3) Else userProfile.setProperty("IS_APPROVED", getIsApprovedPropertyValue()) and call userProfileManager.updateUserProfile(userProfile).

4) return null in case of successful execution.

1.3.5 DomainApprovalRejectionHandler#execute

1) Create the GameData EJB instance [gameData] as described in Common Operations section.
2) Get the domain:
    a) Get requested domain id from request parameter
    String domainId =
    actionContext.getRequest().getParameter(domainIdRequestParamName)
    b) Parse domainId into a long value.
    c) Domain domain = gameData.getDomain(domainId)
3) Search for sponsor
    a) Get sponsor id from request parameter
    String sponsorId =
    actionContext.getRequest().getParameter(sponsorIdRequestParamName)
    b) Parse sponsorId into a long value.
    c) If domain.getSponsorId() does not match with sponsorId, create a HandlerResult instance as :
    new HandlerResult(ResultCode.SPONSOR_NOT_BELONG_TO_DOMAIN, some msg) and set the instance as a request attribute
    d) UserProfile sponsor = userProfileManager.getUserProfile(sponsorId);
    Synchronize over userProfileManager during the method call.
4) If userProfile.getProperty("IS_APPROVED") is not null or blank, create a HandlerResult instance as :
    new HandlerResult(ResultCode.APPROVAL_NOT_PENDING, some msg) and set the instance as a request attribute.
5) Create the AdminData EJB instance [adminData] as described in Common Operations section.
6) Update domain:
    adminData.setDomainApproval(domainId, getApprovedFlag())
7) return null in case of successful execution.

1.3.6 SponsorImageApprovalRejectionHandler#execute

1) Create the GameData EJB instance [gameData] as described in Common Operations section.
2) Get the domain:
    a) Get requested domain id from request parameter
    String domainId =
    actionContext.getRequest().getParameter(domainIdRequestParamName)
    b) Parse domainId into a long value.
    c) Domain domain = gameData.getDomain(domainId)
    d) Get image id from request parameter
    String imageId =
    actionContext.getRequest().getParameter(imageIdRequestParamName)
    e) Parse imageId into a long value.
    f) ImageInfo[] images = domain.getImages(). Iterate through images and check if you find an image with getId() matching imageId, if not found, create a

HandlerResult instance as :
new HandlerResult(ResultCode.IMAGE_NOT_BELONG_TO_DOMAIN, some msg) and set the instance as a request attribute

3) Search for sponsor

   a) Get sponsor id from request parameter
   String sponsorId =
   actionContext.getRequest().getParameter(sponsorIdRequestParamName)

   b) Parse sponsorId into a long value.

   c) If domain.getSponsorId() does not match with sponsorId, create a HandlerResult instance as :
   new HandlerResult(ResultCode.SPONSOR_NOT_BELONG_TO_DOMAIN, some msg) and set the instance as a request attribute

   d) UserProfile sponsor = userProfileManager.getUserProfile(sponsorId parsed into long);
   Synchronize over userProfileManager during the method call.

4) If userProfile.getProperty("IS_APPROVED") is not null or blank, create a HandlerResult instance as :
   new HandlerResult(ResultCode.APPROVAL_NOT_PENDING, some msg) and set the instance as a request attribute.

5) Create the AdminData EJB instance [adminData] as described in Common Operations section.

6) Update image:
   adminData.setImageApproval(imageId, getApprovedFlag())

7) return null in case of successful execution.

1.3.7 PendingWinnerHandler#execute

1) Create the AdminData EJB instance [adminData] as described in Common Operations section.

2) Get the pending winners information :
   PendingWinner[] pendingWinners = adminData.getPendingWinners();

3) Set the information as request attribute
   actionContext.getRequest().setAttribute(pendingWinnerRequestAttrName, pendingWinners)

4) return null in case of successful execution.

1.3.8 PendingWinnerApprovalRejectionHandler#execute

1) Create the AdminData EJB instance [adminData] as described in Common Operations section.

2) Get the pending winners information :
   PendingWinner[] pendingWinners = adminData.getPendingWinners();

3) Get game id from request parameter
   String gameId =
   actionContext.getRequest().getParameter(gameIdRequestParamName)

4) Get user id from request parameter
   String userId =
   actionContext.getRequest().getParameter(userIdRequestParamName)

5) Parse gameId and userId into long values.

6) Verify that the specified user is currently the first pending winner for the specified game. For this iterate through pendingWinners array. If the first element that has matching pendingWinner.getGameId() and gameID, but does not have matching pendingWinner.getPlayerId() and userID, then create a HandlerResult instance : new HandlerResult(ResultCode.WINNER_NOT_FIRST, some msg) and set the instance as a request attribute

7) If match is fine, boolean toApprove = getApprovedFlag(). If toApprove = true, call adminData.approveWinner(winner, new Date()). Else call adminData.rejectWinner(winner)

8) return null in case of successful execution.

1.3.9 GameParameterHandler#execute

1) First check if session is present
   HttpSession session = request.getSession(false);
   If session is null, throw HandlerExecutionException with an error msg.

2) Get ball color id from request parameter
   String ballColorId = request.getParameter(ballColorIdParamName)

3) Parse it into Long value and set the Long value as session attribute
   session.setAttribute(ballColorIdAttrName, ballColorId)

4) Get key count from request parameter
   String keyCount = request.getParameter(keyCountParamName)

5) Parse it into Integer value and set the Integer value as session attribute
   session.setAttribute(keyCountAttrName, keyCount)

6) Get block count from request parameter
   String blockCount = request.getParameter(blockCountParamName)

7) Parse it into Integer value and set the Integer value as session attribute
   session.setAttribute(blockCountAttrName, blockCount)

8) Get start date from request parameter
   String startDt = request.getParameter(startDateParamName)

9) Get date format from request parameter
   String dtFormat = request.getParameter(dtFormatParamName)

10) Parse start date into java.util.Date using dtFormat and set it as session attribute
    session.setAttribute(startDateAttrName, startDt)

11) return null in case of successful execution.

1.3.10 CreateGameHandler#execute

1) First check if session is present
   HttpSession session = request.getSession(false);
   If session is null, throw HandlerExecutionException with an error msg.

2) Get the ball color id:
   Long ballColorId = session.getAttribute(ballColorIdAttrName)

3) Get the key count:
   Integer keyCount = session.getAttribute(keyCountAttrName)

4) Get the block count:
   Integer blockCount = session.getAttribute(blockCountAttrName)

5) Get the game start date:
   Date gameStartDate = session.getAttribute(startDateAttrName)

6) Get the information about the blocks:
   String[] blocks = request.getParameterValues(blockInfoParamName)
7) Initialize JSONObject[] blockObjs = new JSONObject[blocks.length]
8) Initialize JSON decoder:
   JSONDecoder decoder = new StandardJSONDecoder();
9) For every element in blocks
   a) blockObjs[i] = (JSONObject) decoder.decode(blocks[i]);
10) Create the GameData EJB instance [gameData] as described in Common Operations section.
11) Get the BallColor object for the ball color id:
    BallColor[] ballcolors = gameData.findAllBallColors()
    Iterate through ballcolors and find [ballColor] with matching ballColor.getId() and ballColorId.
12) Create a game instance:
    a) GameImpl game = new GameImpl();
    b) game.setBallColor(ballColor)
    c) game.setKeyCount(keyCount.intValue())
    d) game.setStartDate(gameStartDate)
    e) Set hosting blocks for the game:
        i. HostingBlock[] hBlocks = new HostingBlock[blockObjs.length];
        ii. For every [obj] in blockObjs, create a HostingBlockImpl instance:
            HostingBlockImpl newBlock = new HostingBlockImpl();
            newBlock.setMaxHostingTimePerSlot(obj.getInt(maxSlotTimePropName))
            hBlocks[i] = newBlock;
        iii. game.setBlocks(hBlocks);
13) Persist the game:
    Game newGame = gameData.createGame(game)
14) Get date format from request parameter
    String dtFormat = request.getParameter(dtFormatParamName)
15) Get the auction manager instance from application context:
    AuctionManager auctionMgr =
    session.getServletContext().getAttribute(auctionMgrAttrName)
16) Use auction framework to create auctions for each block in newGame. For each HostingBlock [newHBlock] in newGame.getBlocks() and corresponding [obj] in blockObjs:
    a) String summary = "Hosting slots for game <game name>, block <block number>" where <game name> is newGame.getName() and <block number> is newHBlock.getSequenceNumber()
    b) String description = summary;
    c) int itemCount = obj.getInt(slotCountPropName)
    d) String startDate = obj.getString(auctionStartTimePropName)
    e) String endDate = obj.getString(auctionEndTimePropName)

f) Parse startDate and endDate into java.util.Date using dtFormat

g) Auction auction = new AuctionImpl(null, summary, description, itemCount, this.minimumBid, startDate, endDate, new Bid[0])

h) auctionMgr.createAuction(auction)

17) return null in case of successful execution.

1.3.11 RegenerateBrainteaserOrPuzzleHandler#execute

1) Create the GameData EJB instance [gameData] as described in Common Operations section.

2) Get slot id from request parameter
String slotId = request.getParameter(slotIdParamName)
Parse the slotId into a long value.

3) Get the hosting slot
HostingSlot slot = gameData.getSlot(slotId)

4) If slot.getHostingEnd() is a date equal or greater than now, create a HandlerResult instance as :
new HandlerResult(ResultCode.SLOT_FINISHED_HOSTING, some msg) and set the instance as a request attribute.

5) If slot.getHostingStart() is a date equal or before now, create a HandlerResult instance as :
new HandlerResult(ResultCode.SLOT_STARTED_HOSTING, some msg) and set the instance as a request attribute.

6) Get the AdministrationManager instance from application context:
AdministrationManager adminMgr =
session.getServletContext().getAttribute(adminMgrAttrName)

7) boolean generatePuzzle = generatePuzzle();

a) if generatePuzzle = true, generate puzzle:
adminMgr.regeneratePuzzle(slotId)

b) Else Regenerate brain teasers:
adminMgr.regenerateBrainTeaser(slotId)

8) return null in case of successful execution.

1.3.12 ReorderSlotsHandler#execute

1) Create the GameData EJB instance [gameData] as described in Common Operations section.

2) Get game id from request parameter
String gameId =
actionContext.getRequest().getParameter(gameIdRequestParamName)

3) Get slot id from request parameter
String slotId = request.getParameter(slotIdRequestParamName)

4) Parse the gameId and slotId into long values.

5) Get game instance:
Game game = gameData.getGame(gameId)

6) Search for the HostingBlock [block] which contains the given slot id. Use game.getBlocks() and hostingBlock.getSlots() to find a HostingSlot [slot] which matches the input slotId. Also remember the slot's current index in the array of HostingSlot's for the block [currentSlotIndex].

7) If slot.getHostingEnd() is a date equal or greater than now, create a HandlerResult instance as :
new HandlerResult(ResultCode.SLOT_FINISHED_HOSTING, some msg) and set the instance as a request attribute.

8) If slot.getHostingStart() is a date equal or before now, create a HandlerResult instance as :
new HandlerResult(ResultCode.SLOT_STARTED_HOSTING, some msg) and set the instance as a request attribute.

9) Get new slot offset from request parameter
String offset = request.getParameter(offsetRequestParamName)
Parse offset into an int value.

10) If currentIndex + offset >= block.getSlots().length, create HandlerResult instance as:
new HandlerResult(ResultCode.CANNOT_MOVE_SLOT_BEYOND_LAST, some msg) and set the instance as a request attribute.

11) Update the sequence numbers of the slots based on current slot index and offset. For example, if slot with id 3 is to be moved with offset = 2:

Slot id	1	2	3	4	5	6
Old Sequence Number	1	2	3	4	5	6
New Sequence Number	1	2	5	3	4	6

Ids marked in blue represent slots to be updated as a result of this operation.

12) For each of the slots to be updated, we have to create a new HostingSlotImpl instance and:

a) copy all data from corresponding old HostingSlot instance. Use getId(), getDomain(), getImageId(), getBrainTeaserIds(), getPuzzleId(), getDomainTargets(), getWinningBid(), getHostingStart(), getHostingEnd()

b) set the hosting slot's new sequence number

13) collect the updated HostingSlot instances in an array HostingSlot[] newSlots.

14) Update the slots : gameData.updateSlots(newSlots).

15) return null in case of successful execution.

1.3.13 DeleteSlotHandler#execute

1) Create the GameData EJB instance [gameData] as described in Common Operations section.

2) Get game id from request parameter
String gameId =
actionContext.getRequest().getParameter(gameIdRequestParamName)

3) Get slot id from request parameter
String slotId = request.getParameter(slotIdRequestParamName)

4) Parse the gameId and slotId into long values.

5) Get game instance:
Game game = gameData.getGame(gameId)

6) Search for the HostingBlock [block] which contains the given slot id. Use game.getBlocks() and hostingBlock.getSlots() to find a HostingSlot [slot] which matches the input slotId.

7) If slot.getHostingEnd() is a date equal or greater than now, create a HandlerResult instance as :
   new HandlerResult(ResultCode.SLOT_FINISHED_HOSTING, some msg) and set the instance as a request attribute.
8) If slot.getHostingStart() is a date equal or before now, create a HandlerResult instance as :
   new HandlerResult(ResultCode.SLOT_STARTED_HOSTING, some msg) and set the instance as a request attribute.
9) Delete the slot:
   gameData.deleteSlot(slotId)
10) return null in case of successful execution.

1.3.14 AdministrationManager#regeneratePuzzle (slotId, adminData, gameData)

1) Get the hosting slot
   HostingSlot slot = gameData.getSlot(slotId)
2) Get the image for the puzzle:
   DownloadData imageData = gameData.getDownloadData(slot.getImageId())

   com.topcoder.util.image.manipulation.Image image = new MutableMemoryImage(ImageIO.read(imageData.getContent()))
3) Use Math.random() to choose a number between 0 and (PUZZLE_TYPES.length – 1). We have to generate a puzzle of type specified by the corresponding element in PUZZLE_TYPES [chosenPuzzle].
4) Get puzzle config for the type :
   PuzzleConfig config = puzzleConfigMap.get(chosenPuzzle)
   Synchronize over puzzleConfigMap when retrieving the value.
5) Set puzzle generator configuration:
   a) PuzzleType puzzleType = puzzleTypeSource.getPuzzleType(config.getPuzzleTypeName())
   b) PuzzleGenerator puzzleGenerator = puzzleType.createGenerator();
   c) puzzleGenerator.setAttribute(“image”, image);
   d) puzzleGenerator.setAttribute(“width”, config.getWidth());
   e) puzzleGenerator.setAttribute(“height”, config.getHeight());
6) Generate puzzle:
   PuzzleData puzzleData = puzzleGenerator.generatePuzzle()
7) Store the puzzle and get id:
   long[] puzzleIds = adminData.storePuzzles(new PuzzleData[] { puzzleData });
8) Update slot :
   a) Create a new HostingSlotImpl instance [newSlot].
   b) copy all data from the old slot instance. Use getId(), getDomain(), getImageId(), getBrainTeaserIds(), getDomainTargets(), getWinningBid(), getHostingStart(), getHostingEnd(), getSequenceNumber()
   c) Set the new puzzle id : newSlot.setPuzzleId(puzzleIds[0])
   d) gameData.updateSlots(new HostingSlot[] {newSlot})
9) If an exception such as RemoteException or GameDataException occurs, wrap in AdministrationException and re-throw.

1.3.15 AdministrationManager#regenerateBrainTeaser(slotId, adminData, gameData)

1) Get the hosting slot
   HostingSlot slot = gameData.getSlot(slotId)
2) Get text for puzzle:
   DomainTarget[] targets = slot.getDomainTargets();
   String puzzleText = targets[0].getIdentifierText();
3) Use Math.random() to choose a number between 0 and (BRAINTEASER_TYPES.length – 1). We have to generate a puzzle of type specified by the corresponding element in BRAINTEASER _TYPES [chosenPuzzle].
4) Get puzzle config for the type :
   PuzzleConfig config = puzzleConfigMap.get(chosenPuzzle)
   Synchronize over puzzleConfigMap when retrieving the value.
5) Generate the puzzle series:
   a) PuzzleType puzzleType = puzzleTypeSource.getPuzzleType(config.getPuzzleTypeName())
   b) PuzzleGenerator puzzleGenerator = puzzleType.createGenerator();
   c) puzzleGenerator.setAttribute("text", puzzleText);
   d) PuzzleData[] puzzleDatas = puzzleGenerator.generatePuzzleSeries(config.getPuzzleSeriesSize())
6) Store the puzzles and get ids:
   long[] puzzleIds = adminData.storePuzzles(puzzleDatas);
7) Update slot :
   a) Create a new HostingSlotImpl instance [newSlot].
   b) copy all data from the old slot instance. Use getId(), getDomain(), getImageId(), getPuzzleId(), getDomainTargets(), getWinningBid(), getHostingStart(), getHostingEnd(), getSequenceNumber()
   c) Set the new brainteaser ids : newSlot.setBrainTeaserIds(puzzleIds)
   d) gameData.updateSlots(new HostingSlot[] {newSlot})
8) If an exception such as RemoteException or GameDataException occurs, wrap in AdministrationException and re-throw.

1.3.16 AdministrationManager#generateHuntTargets(slotId, adminData, gameData)

1) Get the hosting slot
   HostingSlot slot = gameData.getSlot(slotId)
2) String domainName = slot.getDomain().getDomainName();
3) Spider the domain:
   a) WebCrawler crawler = new WebCrawler();
   b) crawler.getStrategy().addAddress(new WebAddressContext(domainName, 0));
   c) List result = crawler.crawl();
4) Create a SiteStatistics instance:
   a) Create an instance of ObjectFactory.
      Use new ObjectFactory(new ConfigManagerSpecificationFactory(NAMESPACE), ObjectFactory.BOTH);

b) Then use ObjectFactory to create an instance of SiteStatistics [siteStatistics]. Use objectFactory.createObject("SiteStatistics")

5) Feed the pages from crawler to site statistics:

6) Map docUrlMap = new HashMap();
for (int i = 0; i < results.size(); i++) {
WebPageData data = (WebPageData) results.get(i);
if (data.getContentsAsString() != null) {
String docId = “docID” + I;
siteStatistics.accumulateFrom(data.getContentsAsString(),docId);
docUrlMap.put(docId, data.getAddressContext().getURL());
}
}

7) TextStatistics[] stats = siteStatistics.getElementContentStatistics();

8) For cnt = 0 to preferredTargetLength

a) Pick a TextStatistics [textStatistics] at random from stats array, ensuring that it has not already been selected.

b) If textStatistics.getDocuments().length is >= minDistinctPages and <= maxDistinctPages,

i. Select a document [chosenDocId] on which it appears by choosing an element at random from textStatistics.getDocuments().

ii. create a DomainTargetImpl instance [domainTarget], and call domainTarget.setUriPath(docUrlMap.get(chosenDocId)) domainTarget.setIdentifierText(textStatistics.getText());

iii. Collect domainTarget in an array [domainTargets].

iv. Increment cnt++

9) Update slot :

a) Create a new HostingSlotImpl instance [newSlot].

b) copy all data from the old slot instance. Use getId(), getDomain(), getImageId(), getPuzzleId(), getBrainTeaserIds (), getWinningBid(), getHostingStart(), getHostingEnd(), getSequenceNumber()

c) Set the new domain targets: newSlot.setDomainTargets(domainTargets)

d) gameData.updateSlots(new HostingSlot[] {newSlot})

10) If an exception such as RemoteException or GameDataException occurs, wrap in AdministrationException and re-throw.

1.3.17 AdministrationManager#initializeSlotsForBlock(blockId)

1) Create the AdminData EJB instance [adminData] as described in Common Operations section.

2) Create the GameData EJB instance [gameData] as described in Common Operations section.

3) Get the block
HostingBlock block = gameData.getBlock(blockId)

4) For each HostingSlot’s id in block

a. Call regeneratePuzzle(slotId, adminData, gameData)

b. Call regenerateBrainTeaser(slotId, adminData, gameData)

c. Call generateHuntTargets(slotId, gameData)

5) If an exception such as RemoteException or GameDataException occurs, wrap in AdministrationException and re-throw.

1.4 Component Class Overview

package com.orpheus.administration

class AdministrationManager

A class providing an API for administrative functions that need to be accessible to other components or that need to be accessed by some handlers of this component.

class ServiceLocator

This class is an implementation of the Service Locator pattern. It is used to lookup resources such as EJBHome.

package com.orpheus.administration.handlers

class AdminSummaryHandler

Provides a Handler implementation to return administrative summary data. It will assign the AdminSummary to a configurably named request attribute.

class PendingSponsorHandler

Provides a handler implementation that loads overview data about all sponsors pending approval.

class PendingSponsorDomainHandler

Provides a handler implementation that loads detail data about a given domain.

class PendingWinnerHandler

Provides a Handler implementation to return data about pending winners. It will assign the PendingWinner[] to a configurably named request attribute.

class PendingWinnerApprovalRejectionHandler

Provides an abstract base handler implementation that approves or rejects a given pending winner.

class PendingWinnerApprovalHandler

This class extends the abstract class PendingWinnerApprovalRejectionHandler and implements the abstract method getApprovedFlag() to simply return true in order to approve the pending winner.

class PendingWinnerRejectionHandler

This class extends the abstract class PendingWinnerApprovalRejectionHandler and implements the abstract method getApprovedFlag() to simply return false in order to reject the pending winner.

class DomainApprovalRejectionHandler

Provides an abstract base handler implementation that approves or rejects a given domain.

class DomainApprovalHandler

This class extends the abstract class DomainApprovalRejectionHandler and implements the abstract method getApprovedFlag() to simply return true in order to approve the domain.

class DomainRejectionHandler

This class extends the abstract class DomainApprovalRejectionHandler and implements the abstract method getApprovedFlag() to simply return false in order to reject the domain.

class SponsorApprovalRejectionHandler

Provides an abstract base handler implementation that approves or rejects a given sponsor.

class SponsorApprovalHandler

This class extends the abstract class SponsorApprovalRejectionHandler and implements the abstract method getIsApprovedPropertyValue() to simply return "Y" as value to be set for IS_APPROVED property in order to approve the sponsor.

class SponsorRejectionHandler

This class extends the abstract class SponsorApprovalRejectionHandler and implements the abstract method getIsApprovedPropertyValue() to simply return "N" as value to be set for IS_APPROVED property in order to reject the sponsor.

class SponsorImageApprovalRejectionHandler

Provides an abstract base handler implementation that approves or rejects a given image.

class SponsorImageApprovalHandler

This class extends the abstract class SponsorImageApprovalRejectionHandler and implements the abstract method getApprovedFlag() to simply return true in order to approve the image.

class SponsorImageRejectionHandler

This class extends the abstract class SponsorImageApprovalRejectionHandler and implements the abstract method getApprovedFlag() to simply return false in order to reject the image.

class GameParameterHandler

Provides a Handler implementation that accepts general information about a game to be created, and records it in the user's session in configurably-named attributes.

class CreateGameHandler

Provides a Handler implementation that accepts general information about one or more 'blocks' of sites in a game. It then creates a new game object, records it via the GameData EJB, and initiates auctions for the configured blocks via an Auction Manager instance.

class RegenerateBrainteaserOrPuzzleHandler

Provides an abstract base handler implementation that (re)generates brain teasers or puzzle for a given slot id.

class RegenerateBrainteaserHandler

This class extends the abstract class RegenerateBrainteaserOrPuzzleHandler and implements the abstract method generatePuzzle() to simply return false in order to generate brain teasers.

class RegeneratePuzzleHandler

This class extends the abstract class RegenerateBrainteaserOrPuzzleHandler and implements the abstract method generatePuzzle() to simply return true in order to generate puzzle.

class ReorderSlotsHandler

Provides a handler implementation that reorders as-yet unreached slots within the same block by moving one slot to a different position.

class DeleteSlotHandler

Provides a handler implementation that deletes an as-yet unreached slot from among the future slots.

package com.orpheus.administration.entities

class HandlerResult

This class encapsulates the result of a handler operation. It holds a result code, an explanatory message and an exception, if any, which caused the failure.

class ResultCode

This provides an enumeration of result codes denoting the result of a handler operation.

class PuzzleTypeEnum

This provides an enumeration of puzzle types used in this component.

class PuzzleConfig

Encapsulates configuration data for various puzzle types.

class SponsorDomain

This is a data structure which holds a given sponsor represented by UserProfile instance and its domains represented by Domain[].

class GameImpl

The implementation of Game interface from Game Persistence component which represents a BarookaBall game.

class HostingBlockImpl

The implementation of HostingBlock interface from Game Persistence component which represents a "block" of Barooka Ball hosts.

class HostingSlotImpl

The implementation of HostingSlot interface from Game Persistence component representing the persistent information about a particular hosting slot

class DomainTargetImpl

The implementation of DomainTarget from Game Persistence component which Represents an object to be found on an Orpheus host site.

1.5 Component Exception Definitions

AdministrationException

This class acts as the base class for all custom exceptions in this component such as ConfigurationException and ServiceLocatorException. In addition it is also thrown to indicate errors such as remote exception.

ServiceLocatorException

This exception is thrown from the ServiceLocator class to denote errors when creating the InitialContext or when looking up ejb remote home references. It is used to wrap NamingException when performing JNDI lookups.

ConfigurationException

This exception is thrown to indicate a configuration error such as missing properties or missing configuration which prevent initialization of AdministrationManager.

IllegalArgumentException

This system exception has been used in this component to denote null or invalid argument values. It is also used to denote missing configuration data for the handler constructors, to be consistent with existing handler implementations provided with the Front Controller component.

1.6 Thread Safety

This component is thread-safe since it is intended to be used in a web application.

AdministrationManager class is thread-safe. It is immutable. The puzzleTypeSource instance variable is thread-safe by nature. The methods synchronize over puzzleConfigMap whenever it is accessed.

ServiceLocator class is not thread safe. However it is used in a thread safe manner by this component, in that one instance is not shared across method calls.

All the Handler implementations are thread-safe and they achieve the same from immutability. They also synchronize over instance variables wherever required.

The HandlerResult, ResultCode, PuzzleConfig, PuzzleTypeEnum and SponsorDomain classes are also thread-safe since they are immutable.

The GameImpl, HostingBlockImpl, HostingSlotImpl and DomainTargetImpl classes are not thread safe. However it is used in a thread safe manner by this component, in that one instance is not shared across method calls.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- Front Controller 2.1 – to provide Handler implementations
- Type Safe Enum 1.0 – for ResultCode class
- User Profile 1.0 – to represent users
- User Profile Manager 1.0 – to retrieve user.
- Base Exception 1.0 – as base for exceptions
- JSON object 1.0 – to parse json string
- Web Spider 1.0 – to spider a web site
- Web Site Statistics 1.0
- Puzzle Framework 1.0
- Auction Framework 1.0
- Orpheus Administration Persistence 1.0
- Orpheus Game Persistence 1.0
- JNDI Context Utility 1.0 – to look up EJB services.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component

installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- None

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.orpheus.administration

com.orpheus.administration.handlers

com.orpheus.administration.entities

3.2 Configuration Parameters

Note : The <handler> element is common to all the handler configuration. This is in requirement of the design of the Front Controller component. The value of the type attribute need not be as specified in the samples below. It could be anything as long as it matches up with the name attribute “handler-ref” element defined in the configuration files required by Front Controller.

3.2.1 AdminSummaryHandler configuration

A sample handler element for this would look as follows :

```
<handler type="adminSummary">
  <!--  JNDI name to use to lookup the AdminDataHome interface -->
  <admin-data-jndi-name>AdminDataHome</admin-data-jndi-name>

  <!--  name of request attribute to store AdminSummary in -->
  <summary-request-attribute>AdminSummary</summary-request-attribute>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler’s attribute)

```
<!ELEMENT handler (admin-data-jndi-name, summary-request-attribute, fail-result, fail-request-attribute)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
<!ELEMENT summary-request-attribute (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.2 PendingSponsorHandler configuration

A sample handler element for this would look as follows :

```
<handler type="pendingSponsor">
  <!--  Namespace to use when instantiating ConfigManagerSpecificationFactory -->
<object-factory-ns>objFactoryNS</object-factory-ns>

  <!--  JNDI name to use to lookup the GameDataHome interface -->
```

```
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

  <!--  name of request attribute to store pending sponsors and associated domains in -->
  <sponsor-domain-request-attribute>SponsorDomains</sponsor-domain-request-
attribute>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (object-factory-ns, admin-data-jndi-name, sponsor-domain-request-
attribute, fail-result, fail-request-attribute)>
<!ELEMENT object-factory-ns (#PCDATA)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
<!ELEMENT sponsor-domain-request-attribute (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.3 PendingSponsorDomainHandler configuration

A sample handler element for this would look as follows :

```
<handler type="pendingSponsorDomain">
  <!--  Namespace to use when instantiating ConfigManagerSpecificationFactory -->
  <object-factory-ns>objFactoryNS</object-factory-ns>

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

  <!--  name of request parameter which will contain the domain id -->
  <domain-id-request-param>DomainId</domain-id-request-param>

  <!--  name of request attribute to store Domain in -->
  <domain-request-attribute>Domain</domain-request-attribute>

  <!--  name of request attribute to store domain's associated sponsor in -->
  <sponsor- request-attribute>Sponsor</sponsor-request-attribute>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (object-factory-ns, admin-data-jndi-name, domain-id-request-param,
domain-request-attribute, sponsor-request-attribute, fail-result, fail-request-attribute)>
<!ELEMENT object-factory-ns (#PCDATA)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
<!ELEMENT domain-id-request-param (#PCDATA)>
<!ELEMENT domain-request-attribute (#PCDATA)>
<!ELEMENT sponsor-request-attribute (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
```

```
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.4 *SponsorApprovalHandler and SponsorRejectionHandler configuration*

A sample handler element for this would look as follows :

```
<handler type="sponsorApproval">
  <!--  Namespace to use when instantiating ConfigManagerSpecificationFactory -->
  <object-factory-ns>objFactoryNS</object-factory-ns>

  <!--  name of request parameter which will contain the sponsor id -->
  <sponsor-id-request-param>sponsorId</sponsor-id-request-param>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (object-factory-ns, sponsor-id-request-param, fail-result, fail-
request-attribute)>
<!ELEMENT object-factory-ns (#PCDATA)>
<!ELEMENT sponsor-id-request-param (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.5 *DomainApprovalHandler and DomainRejectionHandler configuration*

A sample handler element for this would look as follows :

```
<handler type="domainApproval">
  <!--  Namespace to use when instantiating ConfigManagerSpecificationFactory -->
  <object-factory-ns>objFactoryNS</object-factory-ns>

  <!--  JNDI name to use to lookup the AdminDataHome interface -->
  <admin-data-jndi-name>AdminDataHome</admin-data-jndi-name>

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

  <!--  name of request parameter which will contain the domain id -->
  <domain-id-request-param>DomainId</domain-id-request-param>

  <!--  name of request parameter which will contain the sponsor id -->
  <sponsor-id-request-param>sponsorId</sponsor-id-request-param>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (object-factory-ns, admin-data-jndi-name, game-data-jndi-name,
domain-id-request-param, sponsor-id-request-param, fail-result, fail-request-attribute)>
<!ELEMENT object-factory-ns (#PCDATA)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
```

```
<!ELEMENT game-data-jndi-name (#PCDATA)>
<!ELEMENT domain-id-request-param (#PCDATA)>
<!ELEMENT sponsor-id-request-param (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.6 SponsorImageApprovalHandler and SponsorImageRejectionHandler configuration

A sample handler element for this would look as follows :

```
<handler type="sponsorImageApproval">
  <!--  Namespace to use when instantiating ConfigManagerSpecificationFactory -->
  <object-factory-ns>objFactoryNS</object-factory-ns>

  <!--  JNDI name to use to lookup the AdminDataHome interface -->
  <admin-data-jndi-name>AdminDataHome</admin-data-jndi-name>

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

  <!--  name of request parameter which will contain the image id -->
  <image-id-request-param>ImageId</image-id-request-param>

  <!--  name of request parameter which will contain the domain id -->
  <domain-id-request-param>DomainId</domain-id-request-param>

  <!--  name of request parameter which will contain the sponsor id -->
  <sponsor-id-request-param>sponsorId</sponsor-id-request-param>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (object-factory-ns, admin-data-jndi-name, game-data-jndi-name,
image-id-request-param, domain-id-request-param, sponsor-id-request-param, fail-result,
fail-request-attribute)>
<!ELEMENT object-factory-ns (#PCDATA)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
<!ELEMENT game-data-jndi-name (#PCDATA)>
<!ELEMENT image-id-request-param (#PCDATA)>
<!ELEMENT domain-id-request-param (#PCDATA)>
<!ELEMENT sponsor-id-request-param (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.7 PendingWinnerHandler configuration

A sample handler element for this would look as follows :

```
<handler type="pendingWinner">
  <!--  JNDI name to use to lookup the AdminDataHome interface -->
  <admin-data-jndi-name>AdminDataHome</admin-data-jndi-name>

  <!--  name of request attribute to store PendingWinner[] in -->
  <pending-winner-request-attribute>PendingWinner</pending-winner-request-attribute>
```

```
  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (admin-data-jndi-name, pending-winner-request-attribute, fail-result,
fail-request-attribute)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
<!ELEMENT pending-winner-request-attribute (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.8 PendingWinnerApprovalHandler & PendingWinnerRejectionHandler configuration

A sample handler element for this would look as follows :

```
<handler type="pendingWinnerApproval">

  <!--  JNDI name to use to lookup the AdminDataHome interface -->
  <admin-data-jndi-name>AdminDataHome</admin-data-jndi-name>

  <!--  name of request parameter which will contain the game id -->
  <game-id-request-param>gameId</game-id-request-param>

  <!--  name of request parameter which will contain the user id -->
  <user-id-request-param>userId</user-id-request-param>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (admin-data-jndi-name, game-id-request-param, user-id-request-
param, fail-result, fail-request-attribute)>
<!ELEMENT admin-data-jndi-name (#PCDATA)>
<!ELEMENT game-id-request-param (#PCDATA)>
<!ELEMENT user-id-request-param (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.9 GameParameterHandler configuration

A sample handler element for this would look as follows :

```
<handler type="gameParameter">

   <!--  name of request parameter which will contain the game id -->
  <ballcolor-id-request-param>ballColorId</ballcolor-id-request-param>

  <!-- name of session attribute which will contain the parsed ball color id -->
  <ballcolor-id-session-attr>ballColorId</ballcolor-id-session-attr>

  <!--  name of request parameter which will contain the key count -->
  <key-count-request-param>keyCount</key-count-request-param>

  <!-- name of session attribute which will contain the parsed key count -->
```

```
  <key-count-session-attr>keyCount</key-count-session-attr>

  <!--  name of request parameter which will contain the block count -->
  <block-count-request-param>blockCount</block-count-request-param>

  <!-- name of session attribute which will contain the parsed block count -->
  <block-count-session-attr>blockCount</block-count-session-attr>

  <!--  name of request parameter which will contain the start date -->
  <start-date-request-param>startDate</start-date-request-param>

  <!--  name of request parameter which will contain the date format -->
  <date-format-request-param>dateFormat</date-format-request-param>

  <!-- name of session attribute which will contain the parsed start date -->
  <start-date-session-attr>startDate</start-date-session-attr>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (ballcolor-id-request-param, ballcolor-id-session-attr, key-count-request-param, key-count-session-attr, block-count-request-param, block-count-session-attr, start-date-request-param, date-format-request-param, start-date-session-attr, fail-result, fail-request-attribute)>
<!ELEMENT ballcolor-id-request-param (#PCDATA)>
<!ELEMENT ballcolor-id-session-attr (#PCDATA)>
<!ELEMENT key-count-request-param (#PCDATA)>
<!ELEMENT key-count-session-attr (#PCDATA)>
<!ELEMENT block-count-request-param (#PCDATA)>
<!ELEMENT block-count-session-attr (#PCDATA)>
<!ELEMENT start-date-request-param (#PCDATA)>
<!ELEMENT date-format-request-param (#PCDATA)>
<!ELEMENT start-date-session-attr (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.10 CreateGameHandler configuration

A sample handler element for this would look as follows :

```
<handler type="createGame">

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

<!--  name of request parameter which will contain the block info -->
  <block-request-param>ballColorId</block-request-param>

  <!--  name of request parameter which contain the property name for max slot time -->
  <max-slot-time-prop>maxSlotTime</max-slot-time-prop>

  <!--  name of request parameter which will contain the property name for slot count -->
  <slot-count-prop>slotCount</slot-count-prop>

  <!--  name of request parameter which will contain the property name for auction start time -->
  <auction-start-time-prop>auctionStartTime</auction-start-time-prop>
```

```
  <!--  name of request parameter which will contain the property name for auction end
time -->
  <auction-end-time-prop>auctionEndTime</auction-end-time-prop>

  <!-- name of session attribute which will contain the parsed ball color id -->
  <ballcolor-id-session-attr>ballColorId</ballcolor-id-session-attr>

  <!-- name of session attribute which will contain the parsed key count -->
  <key-count-session-attr>keyCount</key-count-session-attr>

  <!-- name of session attribute which will contain the parsed block count -->
  <block-count-session-attr>blockCount</block-count-session-attr>

  <!--  name of request parameter which will contain the date format -->
  <date-format-request-param>dateFormat</date-format-request-param>

  <!-- name of session attribute which will contain the parsed start date -->
  <start-date-session-attr>startDate</start-date-session-attr>

  <!-- name of application attribute which will contain the AuctionManager -->
  <auction-mgr-app-attr>auctionMgr</auction-mgr-app-attr>

  <!—minimum bid for any new auction -->
  <minimum-auction-bid>auctionMgr</minimum-auction-bid>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (game-data-jndi-name, block-request-param, max-slot-time-prop,
slot-count-prop, auction-start-time-prop, auction-end-time-prop, ballcolor-id-session-attr,
key-count-session-attr, block-count-session-attr, date-format-request-param, start-date-
session-attr, auction-mgr-app-attr, fail-result, fail-request-attribute)>
<!ELEMENT game-data-jndi-name (#PCDATA)>
<!ELEMENT block-request-param (#PCDATA)>
<!ELEMENT max-slot-time-prop (#PCDATA)>
<!ELEMENT slot-count-prop (#PCDATA)>
<!ELEMENT auction-start-time-prop (#PCDATA)>
<!ELEMENT auction-end-time-prop (#PCDATA)>
<!ELEMENT ballcolor-id-session-attr (#PCDATA)>
<!ELEMENT key-count-session-attr (#PCDATA)>
<!ELEMENT block-count-session-attr (#PCDATA)>
<!ELEMENT date-format-request-param (#PCDATA)>
<!ELEMENT start-date-session-attr (#PCDATA)>
<!ELEMENT auction-mgr-app-attr (#PCDATA)>
<!ELEMENT minimum-auction-bid (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.11 RegenerateBrainteaserHandler & RegeneratePuzzleHandler configuration

A sample handler element for this would look as follows :

```
<handler type="regenerateBrainteaser">

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>
```

```
  <!--  name of request parameter which will contain the slot  id -->
  <slot-id-request-param>slotId</slot-id-request-param>

  <!--  name of application context attribute which will have AdministrationManager
instance-->
  <admin-mgr-app-attr>adminMgr</ admin-mgr-app-attr >

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (game-data-jndi-name, slot-id-request-param, admin-mgr-app-attr,
fail-result, fail-request-attribute)>
<!ELEMENT game-data-jndi-name (#PCDATA)>
<!ELEMENT slot-id-request-param (#PCDATA)>
<!ELEMENT admin-mgr-app-attr (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.12 ReorderSlotsHandler configuration

A sample handler element for this would look as follows :

```
<handler type="reorderSlots">

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

  <!--  name of request parameter which will contain the game id -->
  <game-id-request-param>gameId</game-id-request-param>

  <!--  name of request parameter which will contain the slot  id -->
  <slot-id-request-param>slotId</slot-id-request-param>

  <!--  name of request parameter which will contain the new slot offset -->
  <offset-request-param>offset</offset-request-param>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (game-data-jndi-name, game-id-request-param, slot-id-request-
param, offset-request-param, fail-result, fail-request-attribute)>
<!ELEMENT game-data-jndi-name (#PCDATA)>
<!ELEMENT game-id-request-param (#PCDATA)>
<!ELEMENT slot-id-request-param (#PCDATA)>
<!ELEMENT offset-request-param (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.13 *DeleteSlotHandler configuration*

A sample handler element for this would look as follows :

```
<handler type="deleteSlot">

  <!--  JNDI name to use to lookup the GameDataHome interface -->
  <game-data-jndi-name>GameDataHome</ game-data-jndi-name>

  <!--  name of request parameter which will contain the game id -->
  <game-id-request-param>gameId</game-id-request-param>

  <!--  name of request parameter which will contain the slot  id -->
  <slot-id-request-param>slotId</slot-id-request-param>

  <!--  name of result to return in case of execution failure -->
  <fail-result>Failed</fail-result>

  <!--  name of request attribute to store HandlerResult in case of failure-->
  <fail-request-attribute>Failed</fail-request-attribute>
</handler>
```

DTD for this handler element (not including handler's attribute)

```
<!ELEMENT handler (game-data-jndi-name, game-id-request-param, slot-id-request-
param, fail-result, fail-request-attribute)>
<!ELEMENT game-data-jndi-name (#PCDATA)>
<!ELEMENT game-id-request-param (#PCDATA)>
<!ELEMENT slot-id-request-param (#PCDATA)>
<!ELEMENT fail-result (#PCDATA)>
<!ELEMENT fail-request-attribute (#PCDATA)>
```

3.2.14 *AdministrationManager configuration*

Following parameters are expected to be in the namespace used by AdministrationManager.

Property	Value	Description
jigsaw.puzzleTypeName	Name of puzzle type for jigsaw as configured in puzzle framework.	Required.
jigsaw.width	Width in pieces for jigsaw puzzle type.	Required. Must be a positive integer value
jigsaw.height	Height in pieces for jigsaw puzzle type.	Required. Must be a positive integer value
slidingTile.puzzleTypeName	Name of puzzle type for sliding tile as configured in puzzle framework.	Required.
slidingTile.width	Width in pieces for sliding tile puzzle type.	Required. Must be a positive integer value
slidingTile.height	Height in pieces for sliding tile puzzle type.	Required. Must be a positive integer value
missingLetter. puzzleTypeName	Name of puzzle type for missing letter as configured in puzzle framework.	Required.

missingLetter.seriesSize	Size of puzzle series for missing letter type.	Required. Must be a positive integer value
letterScramble. puzzleTypeName	Name of puzzle type for letter scramble as configured in puzzle framework.	Required.
letterScramble.seriesSize	Size of puzzle series for letter scramble type.	Required. Must be a positive integer value
admin-data-jndi-name	the JNDI name to use to look up the AdminDataHome service.	Required.
game-data-jndi-name	the JNDI name to use to look up the GameDataHome service.	Required.
MinTargetLength	Minimum number of targets to generate for any particular slot.	Required. Must be a positive integer value
MaxTargetLength	Maximum number of targets to generate for any particular slot.	Required. Must be a positive integer value
PreferredTargetLength	Preferred number of targets to generate for any particular slot.	Required. Must be a positive integer value
MinDistinctPages	Minimum number of distinct pages that a text should appear to be a candidate for selection.	Required. Must be a positive integer value
MaxDistinctPages	Maximum number of distinct pages that a text should appear to be a candidate for selection.	Required. Must be a positive integer value
PreferredDistinctPages	Preferred number of distinct pages that a text should appear to be a candidate for selection.	Required. Must be a positive integer value

3.3 Dependencies Configuration

See the component specifications of the respective dependencies for configurations required by them.

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

Dependencies should be configured correctly for this component to be used.

4.3 Demo

4.3.1 Handlers demo

To run the handlers, we don't need to write any code. We only need to configure the handlers, which will be called by FrontController to execute.

So the handlers configuration in global-resource.xml could look like below:

<handlers-def>
<handler-def name="adminSummary">com.orpheus.administration.handlers.
AdminSummaryHandler</handler-def>

<handler-def name="pendingSponsor">com.orpheus.administration.handlers.
PendingSponsorHandler</handler-def>

<handler-def name="pendingSponsorDomain">com.orpheus.administration.handlers.
PendingSponsorDomainHandler</handler-def>

<handler-def name="sponsorApproval">com.orpheus.administration.handlers.
SponsorApprovalHandler</handler-def>

<handler-def name="sponsorRejection">com.orpheus.administration.handlers.
SponsorRejectionHandler</handler-def>

<handler-def name="domainApproval">com.orpheus.administration.handlers.
DomainApprovalHandler</handler-def>

<handler-def name="domainRejection">com.orpheus.administration.handlers.
DomainRejectionHandler</handler-def>

<handler-def name="sponsorImageApproval">com.orpheus.administration.handlers.
SponsorImageApprovalHandler</handler-def>

<handler-def name="sponsorImageRejection">com.orpheus.administration.handlers.
SponsorImageRejectionHandler</handler-def>

<handler-def name="pendingWinner">com.orpheus.administration.handlers.
PendingWinnerHandler</handler-def>

<handler-def name="pendingWinnerApproval">com.orpheus.administration.handlers.
PendingWinnerApprovalHandler</handler-def>

<handler-def name="pendingWinnerRejection">com.orpheus.administration.handlers.
PendingWinnerRejectionHandler</handler-def>

<handler-def name="gameParameter">com.orpheus.administration.handlers.
GameParameterHandler</handler-def>

<handler-def name="createGame">com.orpheus.administration.handlers.
CreateGameHandler</handler-def>

<handler-def name=" regenerateBrainteaser">com.orpheus.administration.handlers.
RegenerateBrainteaserHandler</handler-def>

<handler-def name=" regeneratePuzzle">com.orpheus.administration.handlers.
RegeneratePuzzleHandler</handler-def>

<handler-def name="reorderSlots">com.orpheus.administration.handlers.
ReorderSlotsHandler</handler-def>

<handler-def name="deleteSlot">com.orpheus.administration.handlers.
DeleteSlotHandler</handler-def>

</handlers-def>

4.3.2 AdministrationManager demo

Creating a manager instance

PuzzleTypeSource puzzleTypeSource; //configured instance

AdministrationManager mgr = new AdministrationManager(puzzleTypeSource, namespace)

Regenerating a puzzle for a slot id

mgr.regeneratePuzzle(1)

Regenerating brain teasers for a slot id

mgr.regenerateBrainTeaser (1)

Regenerating mini hunt targets for a slot id

mgr.generateHuntTargets (1)

Initializing all slots for a particular block id.

mgr.initializeSlotsForBlock (1)

5. Future Enhancements

Additional handlers could be added. Mini hunt target generation could be enhanced.