Administration Persistence 1.0 Component Specification

1. Design

The Administration Persistence component provides the Orpheus application with an interface to persistent storage of administrative and massage data. The central persistence functionality is handled by stateless session EJBs.

1.1 Design considerations

There are no major considerations. The only issue deals with whether caching would be useful in this component.

1.1.1 Caching considerations

When considering whether caching is feasible, the following factor are considered:

1) The message caching from 1.2.2 is only useful when retrieving, but we only retrieve using search criteria, and unless we are going after something as specific as guid (which is unique), we will always have to query the database, no matter what we have in the cache. In fact, an attempt to match cache and query results is just more expensive that just doing the query. As such, the usefulness of a cache solution here is not feasible. It remains to be seen, after careful analysis of the RSS Generator's SearchCriteria, whether caching based on it, or its elements, will work, once that component is finished.
2) Administration only has two getters, and both are aggregations of data, which will always require us to query the database as the summary will always be instantly stale, and there will new pending winners added or removed, thus making PendingWinner[] data also instantly stale. It is simply not useful to cache aggregated data, because it its inherent volatility. As such, the usefulness of a cache solution here is also not feasible.

As such, caching is currently not incorporated into this design.

1.2 Design Patterns

1.2.1 Data Transfer Object J2EE Pattern

The AdminSummary, PendingWinner, Message, and SearchCriteria can be said to use the **Data Transfer Object** pattern, but their goal is not to improve performance but rather to provide serializability of the data.

1.2.2 Strategy

Used extensively in this design. MessageDAO, AdminDataDAO, MessageTranslator, and SearchCriteriaTranslator interfaces are also part of this pattern. Even the AdminSummary, PendingWinner, Message, and SearchCriteria are used as strategies by the clients, EJBs, and DAOs, although we do provide specific translators for them. Needless to say, the EJB interfaces also use this pattern.

1.2.3 Factory pattern

The DAOFactory uses this pattern, to some degree, to provide a single entry point for EJBs to obtain the MessageDAO and AdminDataDAO, instances In a more orthodox usage of this pattern, the factory would accept a token and retrieve a type of DAO instance based on this token. This factory has a specific method to return DAO of a single type, and they don't manufacture new DAO instances with every call. In reality, the true factory here is the ObjectFactory class.

1.2.4 DAO

The MessageDAO and AdminDataDAO classes use the **DAO** pattern to encapsulate data access in one abstracted place.

1.2.5 Template method

The abstract clients OrpheusMessageDataStore and OrpheusMessengerPlugin implement the work of translating between objects, but leave the details of interfacing with the persistence to the implementations using **Template Method** pattern.

1.3 Industry Standards

JDBC, SQL Server 2000 T-SQL, EJB 2.1

1.4 Required Algorithms

There are no complex algorithms here. Implementation hints are provided in Poseidon documentation as "Implementation Notes" sections.

1.5 Component Class Overview

The component classes and interfaces are spread across three packages. There is the main package where the clients and data access classes are located. The EJBs are located in a separate sub-package, as are the specific data access and translator classes.

1.5.1 com.orpheus.administration.persistence

This is the main package. It contains the clients, data access classes, EJBs, DTOs, and translators are located.

OrpheusMessageDataStore
This is the RSS Generator Datastore persistence client to the EJB layer. It implements the DataStore but supports only RSSItem-related operations. It uses the ConfigManager and Object Factory to initialize itself. It is built to work with EJBs, and this class leaves it to implementations to specify the EJBs. Hence the abstract ejbXXX methods. The public methods defer to these for actual persistence calls. It delegates to MessageTranslator and SearchCriteriaTranslator instances to perform translations between the RSSItem and SearchCriteria instances and their equivalent transport entities – Message and SearchCriteria DTO.

OrpheusMessengerPlugin

This is the Messenger Framework MessengerPlugin persistence client to the EJB layer. It implements the MessengerPlugin and supports its lone operation – send. It uses the ConfigManager and Object Factory to initialize itself. It is built to work with EJBs, and this class leaves it to implementations to specify the EJBs. Hence the abstract ejbXXX methods. The public methods defer to these for actual persistence calls. It delegates to MessageTranslator instance to perform translations between the MessageAPI class, AdminMessage, instance and its equivalent transport entity – Message.

LocalOrpheusMessageDataStore
Implements the abstract ejbXXX methods to work with the local message EJB. Simply defers all calls to the EJB. It uses the ConfigManager and Object Factory to initialize the JNDI EJB reference to obtain the handle to the EJB interface itself.

RemoteOrpheusMessageDataStore
Implements the abstract ejbXXX methods to work with the remote message EJB. Simply defers all calls to the EJB. It uses the ConfigManager and Object Factory to initialize the JNDI EJB reference to obtain the handle to the EJB interface itself.

LocalOrpheusMessengerPlugin
Implements the abstract ejbXXX methods to work with the local message EJB. Simply defers all calls to the EJB. It uses the ConfigManager and Object Factory to initialize the JNDI EJB reference to obtain the handle to the EJB interface itself.

RemoteOrpheusMessengerPlugin
Implements the abstract ejbXXX methods to work with the remote message EJB. Simply defers all calls to the EJB. It uses the ConfigManager and Object Factory to initialize the JNDI EJB reference to obtain the handle to the EJB interface itself.

DAOFactory
Static factory for supplying the DAO instances to the EJBs. It uses synchronized lazy instantiation to get the initial instance of each DAO. Supports the creation of the AdminDataDAO and MessageDAO.

AdminDataDAO
Interface specifying the methods for administration persistence. Supports the RSSItem-related methods, as RSSItem translates to the Message, the DTO used here.

MessageDAO

Interface specifying the methods for message persistence. Works with the Message and SearchCriteriaDTO as the data transfer objects. Supports all methods in the client.

MessageTranslator
Interface specifying the contract for translating between message-related instances and their transport equivalents – Message. The former are value objects used on the outside world and a DTO is an entity this component uses to ferry info between the clients and the DAOs. Implementations will constrain the data types they support.

SearchCriteriaTranslator
Interface specifying the contract for translating between SearchCriteria instances and their transport equivalents – SearchCriteriaDTO. The former are value objects used on the outside world and a DTO is an entity this component uses to ferry info between the clients and the DAOs. Implementations will constrain the data types they support.

AdminDataLocalHome
This is the local home interface for managing administration data. The local client will obtain it to get the local interface.

AdminDataLocal
This the local interface used to talk to the AdminDataBean. Supports all client operations.

AdminDataHome
This is the remote home interface for managing administration data. The remote client will obtain it to get the remote interface.

AdminData
This the remote interface used to talk to the AdminDataBean. Supports all client operations.

AdminDataBean
The EJB that handles the actual client requests. It simply delegates all operations to the AdminDataDAO it obtains from the DAOFactory.

AdminSummary
An interface that represents summary administrative information. It will include the number of pending sponsors and winners, as well as active games. It is used in the DTO in the AdminData EJBs, but the EJBs do not operate on it, instead letting the DAOs work with them.

PendingWinner

An interface that represents the essential details associating a pending game winner with the game he has provisionally won and the payout he will receive if approved. It is used in the DTO in the AdminData EJBs, but the EJBs do not operate on it, instead letting the DAOs work with them.

MessageLocalHome
This is the local home interface for managing messages. The local client will obtain it to get the local interface.

MessageLocal
This the local interface used to talk to the MessageBean.

MessageRemoteHome
This is the remote home interface for managing messages. The remote client will obtain it to get the remote interface.

MessageRemote
This the remote interface used to talk to the MessageBean.

MessageBean
The EJB that handles the actual client requests. It simply delegates all operations to the MessageDAO it obtains from the DAOFactory.

Message
Interface specifying the contract for a message class to be used as the DTO. It will include a guid, category, content and its type, and the time of the message. Implementations will be simple serializable transfer beans. It would transport the data between the client and the DAO layers. It is assembled at both those ends. The EJB layer does not operate on it.

SearchCriteriaDTO
Interface specifying the contract for a search criteria class to be used as the DTO. Its contents will not be know until the RSS Generator component is finished. Implementations will be simple serializable transfer beans. It would transport the data between the client and the DAO layers. It is assembled at both those ends. The EJB layer does not operate on it.

Filter
This interface defines APIs that every implementation must adhere to. It is part of the composite pattern. It is a common interface for both individual and composite components so that both the individual components and composite components can be viewed uniformly. It also defines a set of static final integer constants representing Filter types. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and

cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

1.5.2 *com.orpheus.administration.persistence.impl*

This is the package where the translator, data access and DTO implementations are located.

RSSItemTranslator
Implements MessageTranslator. It translates between the RSSItem and the Message classes. It is plugged into the OrpheusMessageDataStore class to perform these translations. The translation is a simple 1-1 mapping between these entities (as is expected from an implementation of RSSItem).

AdminMessageTranslator
Implements MessageTranslator. It translates between the AdminMessage and the Message classes. It is plugged into the OrpheusMessengerPlugin class to perform these translations. The translation is a simple 1-1 mapping between these entities.

RSSSearchCriteriaTranslator
Implements SearchCriteriaTranslator. Translates from SearchCriteria to SearchCriteriaDTO, so it can be sent to the DAO level. It is plugged into the OrpheusMessageDataStore class to perform these translations. The translation is not known at this time, and will have to wait until RSS generator is complete.

SQLServerAdminDataDAO
Implements AdminDataDAO. Works with SQL Server database and the tables of auction, bid, and effective_bid, hosting_block, download_obj, puzzle, puzzle_attribute, puzzle_resource, game, plyr_compltd_game, image, plyr_won_game, hosting_slot, domain, and sponsor tables. It supports all defined CRUD operations. It uses ConfigManager and Object Factory to configure the connection factory instance. It is expected that the former will use a JNDI connection provider so the Datasource is obtained from the application server. It will create instances of PendingWinner and AdminSummary.

SQLServerMessageDAO
Implements MessageDAO. Works with SQL Server database and the message table. It supports all defined CRUD operations, but the find method is not implemented until RSS generator is complete. It uses ConfigManager and Object Factory to configure the connection factory instance. It is expected that the former will use a JNDI connection provider so the Datasource is obtained from the application server. It will create and consume instances of Message. Uses SearchBundle form the Search Builder component to perform searches in the finder method for messages.

AdminSummaryImpl

Simple implementation of the AdminSummary interface. This class' instances will be created in the SQLServerAdminDataDAO.

PendingWinnerImpl
Simple implementation of the PendingWinner interface. This class' instances will be created in the SQLServerAdminDataDAO.

MessageImpl
Simple implementation of the Message interface. This class' instances will be created in the SQLServerMessageDAO.

SearchCriteriaDTOImpl
Simple implementation of the SearchCriteriaDTO interface. Simply contains the Filter that represents this search.

AdminMessage
Extension of the Messenger Framework's MessageAPI to support message attributes, as defined in the Message interface. All five attributes are mandatory. This class plugs into the OrpheusMessengerPlugin, but is translated to Message there.

1.5.3 com.orpheus.administration.persistence.impl.filter

AbstractAssociativeFilter
This abstract class is a concrete implementer of the Filter interface. It is used to construct associative Filter. This abstract class also provides concrete addFilter() method to add a Filter to the associative Filter. The constructor is protected, since it is only used by its subclasses. The attribute is protected not private, because the subclass of this abstract class should have the access to the attribute. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

AbstractSimpleFilter
This abstract class is a concrete implementer of the Filter interface. It is used to construct simple Filter. This abstract class provides some common functions shared among concrete simple filter classes. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

AndFilter
This class extends AbstractAssociativeFilter. It is used to construct AND search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is

almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

OrFilter
This class extends AbstractAssociativeFilter. It is used to construct OR search criterion. The class is thread safe. The filter attribute is initialized by constructor, and changed by addFilter() method, all other methods use it. Thus it is like a read-write structure. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

NotFilter
This class is a concrete implementer of the Filter interface. It is used to construct NOT search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

InFilter
This class is a concrete implementer of the Filter interface. It is used to construct IN search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

LikeFilter
This filter searches a textual field for a specified sub string that is contained in it. The sub string can appear anywhere, including the beginning, the end and any other places. The filter works for database and LDAP. And also it supports wildcard searching. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed. Four types of value are supported, which are CONTAINS, START_WITH, END_WITH and WITH_CONTENT, all of them are achieved by adding wildcard in proper position.

LessThanFilter
This class extends AbstractSimpeFilter class. It is used to construct lessThan search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The

parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

LessThanOrEqualToFilter
This class extends AbstractSimpeFilter class. It is used to construct lessThanOrEqualTo search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

EqualToFilter
This class extends AbstractSimpeFilter class. It is used to construct EqualTo search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

NullFilter
This class implements the Filter interface and is used to signify a search filter for Null fields. Depending on the search environment, it may be used to search for explicitly Null values, or for search fields when a value is not present. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

BetweenFilter
This class extends AbstractSimpeFilter class. It is used to construct between search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

GreaterThanorEqualToFilter
This class extends AbstractSimpeFilter class. It is used to construct GreaterThanOrEqualTo search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only 1.3 version, and all deprecated items have been removed.

GreaterThanFilter

This class extends AbstractSimpeFilter class. It is used to construct GreaterThan search criterion. It is serializable so it can be used in the Administration EJB framework. It is meant to parallel the Filter in the Search Builder component. The parallel is almost exact, but all implementations will ignore the validation and cloning aspects. It also incorporates only the 1.3 version, and all deprecated items have been removed.

1.6 Component Exception Definitions

This component defines six custom exceptions.

InstantiationException
Extends AdministrationException. This exception is thrown by the constructors of all custom classes in this design that require configuration. The classes that use this exception include: OrpheusMessageDataStore (and its descendants), OrpheusMessengerPlugin (and its descendants), SQLServerAdminDataDAO, SQLServerMessageDAO, and DAOFactory. It is thrown if there is an error during the construction of these objects.

PersistenceException
Extends AdministrationException. This exception is the base exception for persistence operations in this component. As such, it and its three subclasses are thrown by the ejbXXX method in the clients, the business methods in the EJBs, and the DAOs. In effect, the client helper methods, for messages, and EJB business methods act as a pass-though for these exceptions.

DuplicateEntryException
Extends PersistenceException. This is a specific persistence exception when inserting a record with a primary id that already exists. The DAOs and the associated EJBs and, for messages, client helper methods use it.

EntryNotFoundException
Extends PersistenceException. This is a specific persistence exception when retrieving, updating, or deleting a record with a primary id that does not exist. The DAOs and the associated EJBs and, for messages, client helper methods use it.

InvalidEntryException
Extends PersistenceException. This is a specific persistence exception when trying to approve or reject a winner when the action has already been done prior. The AdminData DAO and the associated AdminData EJBs use it.

TranslationException
Extends AdministrationException. This exception is thrown by MessageTranslator and SearchCriteriaTranslator and its subclasses if there is an error while doing the translations. At this point, the implementations do not use it because the translations are very simple and do not involve any checked exceptions.

1.7 Thread Safety

Thread safety is an integral and required part of this component, and this component is indeed almost completely thread-safe.

Most classes are thread-safe because they are immutable. This includes the clients, translators, DAOs, and beans. The EJBs are not thread safe, but the container assumes responsibility for this, and since these are stateless beans, they hold no state for us, so the status of their thread-safety is not a concern of ours. Transaction control, a related topic, is managed by the container.

The DAOFactory synchronizes its methods to avoid synchronization issues with lazy instantiation.

The only class not thread-safe is the AdminMessage, which extends a non-thread-safe MessageAPI. As MessageAPI is external, the only way to make it thread-safe would be to introduce a wrapper. However, it is not expected that more than one thread will operate on an instance at a time, and this class is translated to a Message when it is sent to the client, thus ensuring it is used in a thread-safe manner.

Eve if SearchBuilder 1.3 is used, thread safety will not be imperiled, since SearchCriteria will be remapped into a SearchCriteriaDTO that will only be used inside the client, in a thread-safe manner.

2. Environment Requirements

2.1 Environment

JDK 1.4, J2EE 1.4

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
    - Used for configuration in constructors throughout this component to instantiate required classes and obtain other parameters.
- Object Factory 2.0
    - Used to instantiate classes in constructors throughout this component to instantiate required classes.
- RSS Generator 2.0
    - Provides the DataStore interface as well as the RSSItem and SearchCriteria interfaces. This component implements this persistence interface.
- Orpheus Administration Logic 1.0
    - The main client of this component.
- Search Builder 1.3

- Used to perform searches for messages.

- Messenger Framework 1.0
    - Provides the MessengerPlugin class as well as the MessageAPI class. This component extends these to give the ability for this framework to store messages.
- Puzzle Framework 1.0
    - Provides the PuzzleData interface.
- DBConnection Factory 1.0
    - Provides a convenient access to Connections from a Datasource obtained from JNDI. The implementing DAO classes thus don't have to code their own access to this container resource.

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

There are no third party components that need to be used directly by this component.

3. Installation and Configuration

3.1 Package Names

com.orpheus.administration.persistence
com.orpheus.administration.persistence.impl

3.2 Configuration Parameters

3.2.1 OrpheusMessengerPlugin

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required	Example: "com.topcoder.specify"
messageTranslatorKey	Key for the MessageTranslator to pass to ObjectFactory. Required	Valid key
jndiEjbReference	The JNDI reference for the EJB. Required	Example: "java:comp/env/ejb/MessageLocal"

3.2.2 OrpheusMessageDataStore

Parameter	**Description**	**Details**
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required	Example: “com.topcoder.specify”
messageTranslatorKey	Key for the MessageTranslator to pass to ObjectFactory. Required	Valid key
searchCriteriaTranslatorKey	Key for the SearcCriteriaTranslator to pass to ObjectFactory. Required	Valid key
jndiEjbReference	The JNDI reference for the EJB. Required	Example: “java:comp/env/ejb/ MessageLocal”

3.2.3 DAOFactory

Parameter	**Description**	**Sample Values**
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required	Example: “com.topcoder.specify”
adminDataDAO	Key for the AdminDataDAO to pass to ObjectFactory. Required	Valid key
messageDAO	Key for the MessageDAO to pass to ObjectFactory. Required	Valid key

3.2.4 SQLServerMessageDAO

Parameter	Description	Sample Values
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required	Example: “com.topcoder.specify”
factoryKey	Key for the DB Connection Factory to pass to ObjectFactory. Required.	Valid key
name	Name of the connection to the persistence to get from the DB Connection Factory. Optional.	“myConnection” Will use the factory’s default connection, if available, if name not given.
searchNamespace	Namespace to pass to the SearchBundleManager Required.	Valid namespace with relevant search bundles configured.
searchBundle	The name of the SearchBundle to retrieve from the SearchBundleManager. Required	A valid name that exists in the SearchBundleManager.

3.2.5 SQLServerAdminDataDAO

Parameter	Description	Sample Values
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required	Example: “com.topcoder.specify”
factoryKey	Key for the DB Connection Factory to pass to ObjectFactory. Required.	Valid key
name	Name of the connection to the persistence to get from the DB Connection Factory. Optional.	“myConnection” Will use the factory’s default connection, if available, if name not given.
adminFee	The administration fee as a percentage fraction of winnings. Required. Must be 0.0 – 1.0, inclusive.	0.15

3.3 Dependencies Configuration

3.3.1 DBConnectionFactory, ConfigManager, ObjectFactory, Search Builder

The developer should refer to the component specification of these components to configure them. The later two are used extensively.

3.3.2 DDL for tables

Please see section 1.2.6 of the Requirements Specification.

3.3.3 EJB deployment descriptor

This is provided in the ejb-jar.xml file in the /docs/mappings directory.

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

Both client and EJBs must be deployed in a container that supports EJB 2.1 standard. The client can be deployed in a thinner environment as long as it has access to the necessary J2EE packages and has access to the Home and Remote interfaces and implementations in the classpath. If using local access, it must be collocated with the EJBs. Overall, this just corresponds to standard EJB usage.

The tables defined in the Requirements Specification must exist, and the database must be running. The ObjectFactory and ConfigManager components must be configured properly.

4.3 Demo

See 4.2 above on the required steps to set up the environment. The first part of the demo will only focus on the true public message API, which is the clients. The EJBs will not be shown because they are not expected to be used directly. Furthermore, their usage would be very similar to the clients, as the methods are almost completely parallel.

We will, however, show the AdminData EJBs in action, as this component does present them as the public API for administrative data manipulation.

In both cases, we will only focus on the remote implementations, with the knowledge that usage of the local implementation is the same. We will show method calls directly.

Please note that the search criteria will be demonstrated only in a shallow way, since the RSS generator is not complete yet. It will be demonstrated with the understanding reached in section 1.1.

4.3.1 *Typical DataStore client usage*

```
// create remote instance with a namespace
OrpheusMessageDataStore client = new
RemoteOrpheusMessageDataStore(“myNamespace”);

// we might begin by creating a new RSSItem
RSSItem item1 = some message item with guid: 1
RSSItem item2 = another message item with guid: 2
client.createItem(item1);
client.createItem(item2);
// at this point, both are inserted

// we now update an item, after we make some changes (not shown)
client.updateItem(item2);

// we want to delete an item
client.deleteItem(item1);
// item deleted from persistence

// The find method uses SearchCriteria, and at this time
// it is not possible to accurately demonstrate it, but we can
// still make some assumptions:
// retrieve all items with timestamp between 1/1/2001 and 31/12/2001
Date startDate = date representing 1/1/2001
Date endDate = date representing 31/12/2001
SearchCriteria criteria = criteria using the above dates for comparison
RSSItem[] itemsIn2001 = client.findItems(criteria);
```

4.3.2 *Typical MessagePlugIn client usage*

```
// create remote instance with a namespace
OrpheusMessengerPlugin client = new
RemoteOrpheusMessengerPlugin(“myNamespace”);

// we simply send a AdminMessage
AdminMessage mesage1 = some admin message item with guid: 1
client.sendMessage(message1);
// at this point, this message is persisted
```

4.3.3 *Typical AdminData usage*

This demo shows usage of the EJB. We will demonstrate the use of the bean via its remote interface.

```
// obtain remote interface: the details are omitted.
AdminData admin = remote interface to AdminDataBean

// we might begin by storing some new puzzles
PuzzleData[] puzzles = array of puzzles
Long[] ids = admin.storePuzzles(puzzles);
// at this point, all puzzles are persisted

// we have some images and domains to approve and reject
long imageId1 = an existing image Id: 1
long imageId2 = an existing image Id: 2
long domainId1 = an existing domain Id: 1
long domainId2 = an existing domain Id: 2

admin.setDomainApproval(domainId1, true); // accept this domain
admin.setDomainApproval(domainId2, false); // reject this domain
admin.setImageApproval(imageId1, true); // accept this image
admin.setImageApproval(imageId2, false); // reject this image
```

```
// we now get some information on pending winners and sponsors and active
// games by getting a summary
AdminSummary summary = admin.getAdminSummary();

// we now get some detailed information on all pending winners
PendingWinner[] pendingWinners = admin.getPendingWinners();

// We can approve and reject some pending winners form above
PendingWinner pendingWinner1 = designated winner of a game from above
PendingWinner pendingWinner2 = rejected winner of a game from above

admin.approveWinner(pendingWinner1, new Date());
admin.rejectWinner(pendingWinner2);
```

5. Future Enhancements

1) None at this time.