TC Bulletin 2.0 Component Specification

1. Design

The proliferation of user created content on the internet has lead to the desire for more conversational style interfaces in posting messages to online web pages. This differs from the typical message forum, in that it shares more similarity with IRC than it does a Usenet topic. This format, while not overly conducive to storing and categorizing information, can be much more accessible to those who would otherwise avoid communicating online. By providing individual channels for each user, the barrier of entry can be lowered further still. The main entry of the component is MessageBoard class. This class has a persistence field that is used to manage threads, messages and responses in a persistent storage. In this version an implementation of the persistence interface is provided. This implementation uses Informix 7.3 as a persistent storage. The structure of the database is provided in tc_bulletin.ddl file from /docs directory.

1.1 Design Patterns

Strategy – Different implementation of MessageBoardPersistence class can be plugged into MessageBoard class.
DAO – implemented by the persistence class.

1.2 Industry Standards

None.

1.3 Required Algorithms

None.

1.4 Component Class Overview

MessageBoard
This class is the main class of the component. It has a field of MessageBoardPersistence type used to work with the persistence storage. It has other two fields that indicate the maximum size for a message and the number of messages the persistence should return for a request. It has various methods that allows a user to manipulate threads, messages and responses.
This class it is immutable but it is not completely thread safe. The persistence is not guaranteed to be thread safe. Also this class works with entities which are not thread safe.

MessageBoardPersistence
This interface defines the contract for implementations that will manage threads, messages and responses into a persistent storage. The interface defines various methods that allows a user to create threads, to post messages and responses, to retrieve threads and responses, to update threads, messages and responses, to remove threads messages and responses.
Implementations are not required to be thread safe.

InformixMessageBoardPersistence
This class implements MessageBoardPersistence interface and uses Informix 7.3 as persistent storage. For id generation it uses ID Generator component. To get a number of rows starting from a specified index this class uses rowid function of Informix. To obtain a connection to the database, this class uses Connection Factory component. For the methods that modify the database, transactions are used. The structure of the database is provided in tc_bulletin.ddl file.
This class can be considered as thread safe since DBConnectionFactory implementations and IDGenerator implementations are expected to be thread safe and

transactions are employed for methods that modify the database. Concurrency issues may appear if the database is accessed outside of instances of this class.

ErrorMessagesCache

This class holds the error messages that will be used in MessageBoard class and in the persistence implementation class. It defines four constants under which generic messages can be registered in the map. Generic messages can also be registered in the map under different keys, not defined as constants.

It is not thread safe because access to the errorMessages map is not synchronized.

Thread

This class represents a thread. It extends from BasicEntityData abstract class. It has an id, inherited from the abstract class, a user key and a list of messages that belong to the thread. It provides various methods that allow the user to manage the messages of the thread. The messages can have the id not set (<0) meaning that these messages does not exist in persistence (if the thread is updated in persistence then the new messages will have the ids set).

It is not thread safe because it is mutable.

Message

This class represents a message that is posted on a thread. It extends from CommonEntityData. In addition it has a map of attributes and a response.

This class is not thread safe because it is mutable.

Response

This class represents a response to a message. It extends CommonEntityData abstract class without adding any new fields.

This class is not thread safe because it is mutable (the abstract class is mutable).

MessageAttribute

This class represents a message attribute. A Message instance can have many such attributes. A message attribute has a name and a value.

This class is not thread safe because it is mutable.

BasicEntityData

This abstract class holds the id for concrete classes which extend the this abstract class. It has a setter and a getter for the id field.

This class is not thread safe because it is mutable.

CommonEntityData

This abstract class holds the common fields of the Response and Message classes. It extends from BasicEntityData.The common fields of these two classes are the name, the date of posting and the message.

This class it is not thread safe because it is mutable.

1.5 Component Exception Definitions

MessageBoardManagementException

This is the base custom exception defined in this component. All the other custom exceptions extend from this one. This exception is not thrown directly by any class.

MessageBoardPersistenceException

This is a custom exception that will be thrown in case errors occur when accessing the persistence.

EntityNotFoundException
This is a custom exception that will be thrown if an entity is not found in the persistence.

MessageBoardConfigurationException
This is a custom exception that will be thrown if errors occur when reading the configured properties or if any required property is missing.

1.6 Thread Safety

The component is not thread safe. Thread, Message, MessageAttribute and response are mutable and not thread safe. MessageBoard class it is immutable but it is not completely thread safe because the persistence is not guaranteed to be thread safe; also the entities that this class works with are mutable. The current persistence implementation can be considered as thread safe since DBConnectionFactory implementations and IDGenerator implementations are expected to be thread safe and transactions are employed for methods that modify the database. Concurrency issues may appear if the database is accessed outside of instances of this class.Since thread safety was not required, it makes no sense to synchronize the methods of the MessageBoard; this will only add an unnecessary overhead.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5
- Compile target: Java 1.5

2.2 TopCoder Software Components

Base Exception 2.0 – custom exceptions inherit from BaseCriticalException from this component.
Configuration API 1.0 – user for configuration purposes
Configuration Persistence 1.0 – used to get the configuration from file
Connection Factory 1.1 – used to obtain a connection to the database
Object Factory 2.0.1 – used to create objects
Object Factory Configuration API Plugin 1.0 – used with Object Factory component to create objects
ID Generator 3.0 - used to generate ids.

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.

3. Installation and Configuration

3.1 Package Name

com.topcoder.messaging
com.topcoder.messaging.persistence

3.2 Configuration Parameters

Configuration parameters for MessageBoard class:

Parameter	Description	Values
max_message_size	The maximum size of a message. **Optional.**	any long > 0
message_count	The maximum number of messages to be returned for a request. **Optional.**	any long > 0
object_factory_child_name	Used to get the configuration child necessary for ObjectFactory. **Required.**	non empty string
persistence_key	The key used with ObjectFactory to create a MessageBoardPersistence implementation. **Required.**	non empty string
error_messages_child_name	Used to get the configuration child that holds the generic error messages.If not present it means that there are no generic messages. **Optional.**	non empty string

Note that the child that holds the error messages can have any number of configured properties.

Configuration parameters for InformixMessageBoardPersistence

Parameter	Description	Values
connection_name	The name used to obtain a connection. **Required.**	non empty string
id_generator_name	The name passed to the IDGeneratorFactory to create a IDGenerator implementation. **Required.**	non empty string

3.3 Dependencies Configuration

None.

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

See demo.

4.3 Demo

```
//create a MessageBoard instance
MessageBoard mb = new MessageBoard(configuration);
```

```
//create a new thread; the id will not be set; it wll be generated by the
//persistence when the thread will be persisted
Thread thread = new Thread(“someKey”);


//create the thread into persistence; the thread has no messages
thread = mb.createThread(thread);

//the thread now has an id that was generated by the persistence.

//adding a message to the created thread, in persistence, can be done in
//two ways; one is to add a message to the thread and then update the
//thread, the other way is to use the id of the thread and call a method
//that takes the thread id and the message as parameters.

//create a Message without setting the id
Message message1 = new Message(name,date,message);
//the created message has no response yet
//add an attribute to the message
MessageAttribute ma = new MessageAttribute(“someName”,”someValue”);
message11.addAttribute(ma);

//add the message to the thread (post the message)
mb.postMessage(thread.getId(),message1);
//the thread created above now has a message

//create another Message without setting the id
Message message2 = new Message(name,date,message);
//the created message has no response yet

//add the message to the thread
boolean added = thread.addMessage(message2);

//update thread in persistence
mb.updateThread(thread);
//the thread created in this demo now has two messages without responses

//create a response without setting the id
Response response = new Response(name,date,message);
//add the response to the message1 (post response)
mb.postResponse(message1.getId(),response);
//message1 now has a response

//add some more messages to the thread…

//get the first ten messages from the thread created in this demo
List<Message> messages = mb.getMessages(thread.getId(),0,9);

//get the first ten messages from the thread created in this demo using
//the user key
List<Message> messages = mb.getMessages(thread.getUserKey(),0,9);

//update a message
message1.addAttribute(“name”,”value”);
mb.updateMessage(message);

//update the response of message1
Response response = message1.getResponse();
response.setName(“other name”);
mb.updateResponse(response);

//remove the response of the message1
mb.removeResponse(response.getId());
```

```
//remove message1
mb.removeMessage(message1.getId());
```

5. Future Enhancements

Different implementations of MessageBoardPersistence.