Sales IM Messenger Component Specification

1. Design

The Sales IM Messenger component is part of a larger client-server chat application. It distributes and routes all kinds of messages that are involved. Since connection is not persistent between client and server, messages are pushed to message pools for clients to pull. Additional formatting will be applied on normal chat messages. Six types of messages are derived on top of the abstract Message class. Each message type has its own XML representation.

1.1 Design Patterns

MessageTracker, UserIDRetriever and their implementation implement the strategy pattern for MessengerImpl.

1.2 Industry Standards

JDBC

XML

1.3 Required Algorithms

1.3.1 Load the ChatMessageFormatter by role name

This algorithm is used by FormatterLoader to load the ChatMessageFormater from the configuration file. The sample configuration file locates in docs/

```
Impl note:

1. For the role name, construct a namespace like
com.cronos.im.messenger.FormatterLoader.roleName.

2. Load the user name formatting configuration values from
com.cronos.im.messenger.FormatterLoader.roleName.

2.1 Load all the font attributes from user_name_format.xxx property.
2.2 Create XmlTagFormatter with "font" tag, and add these static attributes

3. Load the timestamp formatting configuration values from
com.cronos.im.messenger.FormatterLoader.roleName. Create XmlTagFormatter
in the similar way as step 3.

4. Load the chat dynamic attributes and pattern from chat_text_dynamic_format
property. Create XmlTagFormatter with "a" tag, pattern and the dynamic
attributes. Create Matcher, create ContentHighligthter, add the formatter to
matcher, and then add the Matcher to ContentHigglighter.

5. Create another XmlTagFormatter with "font" tag.

5.1 Load all the static attributes under chat_text_static_format property,
and add them to the formatter.
5.2 Format the chat text against this formatter. Note that the chat text is
the output of the step 5.

Note: please see the sample configuration file for the output of the formatted
chat text.
```

```
6. Create the ChatMessageFormatterImpl with the arguments in the above steps.
Note that the pattern is loaded from the chat_text_dynamic_format property
in step 4.
```

1.3.1 Format the timestamp against the DateFormatContext

The format context information is retrieved from DateFormatContext. In the version 1.0, three attributes will be retrieved:

Date format: retrieved by key DateFormatContext#DATE_FORMAT_KEY, if not found, the default value is hh:mm:ss a z.

Time zone: retrieved by key DateFormatContext#TIME_ZONE_KEY. If not found, the default value is TimeZone.getID();

Locale: retrieved by key DateFormatContext#TIME_ZONE_KEY. If not found, the default value is Locale.getDefault().getLanguage().

The timestamp will be formatted in this way:

// locale is retrieved by locale language from dateFormatContext

DateFormat format = new SimpleDateFormat(dateFormat, locale);

// timeZone is retrieved by time zone id from dateFormatContext

format.setTimeZone(timeZone);

format.format(timestamp);

1.4 Component Class Overview

Messenger:

This interface is the main entry of this component. It defines the contract of routing the messages to the message pools of the desired recipients.

The implementation of this interface is required to be thread safe.

MessengerImpl:

The MessengerImpl is responsible for routing the messages to the message pools of the desired recipients. Using the Chat Message Pool component, users will be able to receive the messages by pulling them of. All the chat messages that are post to the user session pool are tracked.

Though some inner variables(like ChatContactManager) is not immutable, all the methods of this class can be invoked in multi-thread environment since all of them are synchronized.

XMLMessage:

XMLMessage is the base class of all the other six messages defined in this component. It provides a method to set the date time format which will be used to format the timestamp.

This class is not thread safe since it’s mutable.

ChatMessage:

This is the message typed by a user in a chat room. It should include:

User profile of the person who typed the text (only username is required in XML).

Chat text.

The xml string representation of this message has an extra element to keep the formatted chat text.

The formatted chat text takes the following standard format:

[Name]: [Timestamp]: [Text Message]

Name: the user name

Timestamp: the message timestamp. The format for the time should be configurable

with the default being the following format: hh:mm:ss PM Time zone. For example, 05:34:30 PM EST

Text Message: the chat text

The font and color for each of the above parts in configurable. In particular, the color for text message is different for client and manager. This allows easier differentiation between the two. For now, assume the “Role” property returns either “Client” or “Manager”.

Furthermore, if a text message includes a link to a URL, then the URL text will be displayed as a hyperlink. Below are the rules for determining if text should be generated as a hyperlink:

If the text begins with a http://, https://, ftp:// (not case sensitive)

If the text follows the pattern: www.[Text].[Top Level Domain], where [text] section may include dots, and [Top Level Domain] is defined in http://data.iana.org/TLD/tlds-alpha-by-domain.txt

This class is not thread safe since it’s mutable.

SessionUnavailableMessage:

This is the message to notify a user that the session is unavailable. The sender of this message must be the system.

This class is not thread safe since it’s mutable.

PresenceMessage:

This is the message to indicate if a user is present or absent. Besides the attributes defined by XMLMessage, it includes:

User profile attributes of the person who changed the presence (only username is required in XML)

Absence / Presence

This class is not thread safe since it’s mutable.

AskForChatMessage:

This is the message to ask a user for chatting in a session created by another user. The sender of this message must be the system. Besides the attributes defined by XMLMessage, it includes:

User profile attributes of the person who created the chat

Timestamp of when the session is created

Timestamp of which the user needs to acknowledge

This class is not thread safe since it’s mutable.

EnterChatMessage:

This is the message to signal a user to enter a chat session, after he has accepted for chatting. The sender of this message must be the system.

This class is not thread safe since it's mutable.

MessageTracker:

This interface defines the contract of tracking the chat message. The tracking information could be stored in the database, xml file, etc.

The implementation of this interface is required to be thread safe.

ChatMessageTrackerImpl:

ChatMessageTrackerImpl is used to store the tracking information in the database. It's called by MessengerImpl when a chat message is post to user's session pool.

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

UserIDRetriever:

This interface defines the contract of retrieve the long user id from the user object.

The implementation of this interface is required to be thread safe.

UserIDRetrieverImpl:

This class assumes that the user object is of Long type. If not, throw UserIDRetrieverException.

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

ChatMessageFormatter:

This interface defines the contract of formatting the message from user name, timestamp, and chat text.

The implementation of this interface is required to be thread safe.

ChatMessageFormatterImpl:

ChatMessageFormatterImpl uses the ContentHighlighter to format the chat message.

The instance of this class will be created by FormatterLoader, and used by ChatMessage to format the chat message.

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

FormatterLoader:

FormatterLoader is used to load the ChatMessageFormatter instance from the configuration file and associates them with role name for the easy access.

This class is thread safe since it's properly synchronized..

DateFormatContext:

DateFormatContext holds the date format attributes like date pattern, time zone, locale, etc which will be used by XMLMessage to format the timestamp.

This class is not thread safe since it's mutable.

1.5 Component Exception Definitions

IllegalArgumentException

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

IllegalStateException

This exception is thrown from Message class if any field is missing during formatting the xml string.

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

MessengerException

This exception will be thrown by MessengerException and its implementations if any error occurs during the routing the messages

MessageTrackerException

This exception will be thrown by MessageTracker and its implementation if any error occurred during tracking the chat message

UserIDRetrieverException

This exception will be thrown by UserIDRetriever and its implementation if any error occurred during retrieving the user id.

ChatMessageFormattingException

This exception will be thrown by FormattedChatMessage if any error occurred during formatting the chat message.

FormatterConfigurationException

This exception will be thrown by FormatterLoader if any error occurred during loading the configuration values.

1.6 Thread Safety

Messenger and MessengerImpl are the main classes of this component. MessengerImpl itself is not thread safe, since it contains mutable variables like MessagePool and ChatContactMessage. However, as all the methods of this class are synchronized, application users can still invoke these methods in a multi-thread environment.

The XMLMessage and its six sub classes are mutable and not thread safe. To be thread safe, their set and get methods should be synchronized externally.

FormatterLoader is thread safe, since it's properly synchronized.

ChatMessageFormatter and its implementation are thread safe.

2. Environment Requirements

2.1 Environment

Java 1.4 or higher.

2.2 TopCoder Software Components

Base Exception 1.0

The custom exception extends BaseException in this component.

Configuration Manager 2.1.5

It will be used to load the configuration values.

DB Connection Factory 1.0

It will be used to get named connection.

Chat Contact Manager 1.0

It's used to get the blocked users..

Chat Message Pool 1.0

It's used to post the message to the user pool or session pool

Chat Session Manager 1.0

ChatSession come from this component.

Chat User Profile 2.0

ChatUserProfile class comes from this component

Content Highlighter 1.1

It's used to format the user name, timestamp and chat text.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.cronos.im.messenger

com.cronos.im.messenger.impl

3.2 Configuration Parameters

Configuration for com.cronos.im.messenger.FormatterLoader namespace

Parameter	Description	Values
role_name_key	The property name to get	role

	the role name . **Required**.	
user_name_key	The property name to get the user name . **Required**	tc

Assume that the role name is manager
Configuration values for com.cronos.im.messenger.FormatterLoader.manager

Parameter	Description	Values
user_name_format	The property contains the sub properties representing the font attributes like face, color, size etc. **Required**.	None
user_name_format.xxx	This property contains the attribute for font tag. The name of the property will be face, color, size, etc **Required**.	#2248dd(if property name is color)
timestamp_format	The property contains the sub properties representing the font attributes like face, color, size etc. **Required**.	None
timestamp_format.xxx	This property contains the attribute for font tag. The name of the property will be face, color, size, etc **Required**.	5(if the property name is size)
chat_text_format	The property contains the sub properties representing the font attributes like face, color, size etc. **Required**.	none
chat_text_static_format	The property contains the sub properties representing the font attributes like face, color, size etc. **Required**.	none
chat_text_static_format.xxx	This property contains the attribute for font tag. The name of the property will be face, color, size, etc **Required**.	Arial if property name is face
chat_text_dynamic_format	The property contains the sub properties representing dynamic attribute for XmlTagFormatter. The value is the pattern to find	^http://.*\n

	out the url. **Required**.	
chat_text_dynamic_format.xxx	This property contains the dynamic attributes. **Required**.	http://en.wikipedia.org/wiki/$1

3.3 Dependencies Configuration

ConfigurationManager and DBConnectionFactory should be properly configured to make this component work.

4. Usage Notes

4.1 Required steps to test the component

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

4.2 Required steps to use the component

Preload the configuration file into Configuration Manager. Follow demo.

4.3 Demo

The configuration file listed in the section 3.2 is used in the demo.

4.3.1 Use Message

```
// create the ChatMessage
ChatMessage msg = new ChatMessage();

// set the msg attributes
msg.setXXX(…)

// create the date format context
DateFormatContext context = new DateFormatContext();
String xmlString = msg.toXMLString(context);
System.out.println(xmlString);

// the other five message works in the similar way.
```

4.3.2 User MessengerImpl

```
// create ChatMessageTracker, assume that dbFactory is created
// somewhere
MessageTracker tracker = new ChatMessageTracker(dbFactory,"conn","user");

// create the MessengerImpl. Assume that the MessagePool and
// ChatContactManager are created somewhere.
Messenger messenger = new MessengerImpl(pool,tracker,new
            UserIDRetrieverImpl(),manager);

// post the message.
// assume that userId, sessionId, and session are already defined
messenger.postMessage(msg,userId,sessionId);
messenger.postMessageToAll(msg,session);
messenger.postMessageToOthers(msg,session);
// nothing happens, since msg is of ChatMessage type.
```

```
messenger.postMessage(msg,userId);

// get the MesssagePool
MessagePool pool = messenger.getMessagePool();

// pull the message from pool
Message[] ms1 = pool.pull(userId);
Message[] ms2 = pool.pull(userId,sessionId):
```

5. Future Enhancements

None