ICEmobile : Cloud Push Chat Tutorial

This page last changed on Jun 20, 2013 by tyler.johnson.

Cloud Push Chat Tutorial

The goal of this tutorial to is to create a chat application that uses Cloud Push to deliver chat messages between users in a chat room. If a chat message cannot be delivered directly to the UI, an email alert will be sent to the user.

Cloud Push delivers real time notifications to mobile devices, alerting the user of some state change of interest that has occurred in the network cloud. For additional details on the Cloud Push architecture please see the following guides:

- Cloud Push Documentation
- Cloud Push Overview WIKI

Cloud Push Chat

The full source can be found at the bottom of the tutorial here.

Join Chat Room

We will prompt the user to join our chat room by adding a <mobi:inputText> bound to a User object. Once joined, we un-render these fields by setting hasJoined to true which is bound to the rendered attribute of our <mobi:fieldSetRow>. Please add the following to your page:

```
<mobi:fieldsetGroup>
    <mobi:fieldsetRow rendered="#{!user.hasJoined}">
        <mobi:inputText value="#{user.username}" />
        <mobi:commandButton value="Join Chat" actionListener="#{user.join}" />
    </mobi:fieldsetRow>
</mobi:fieldsetGroup>
```

Send Message

After the user has joined the chat room, a <mobi:inputText> and <mobi:commandButton> will be rendered that will allow the user to send a chat message to other users. Please add the following below the 'join' <mobi:fieldsetRow>:

```
<mobi:fieldsetRow rendered="#{user.hasJoined}">
```

```
        <h:outputText value="Welcome #{user.username}" />
        <br />
        <mobi:inputText value="#{user.message}" />
        <mobi:commandButton value="Send Message" actionListener="#{user.sendMessage}" />
    </mobi:fieldsetRow>
```

Display Chat Messages

Messages posted in the chat room will be displayed using <mobi:outputList> and <ui:repeat> tags. The <ui:repeat> will iterate over the list of sent messages and render each as a <mobi:outputListItem>. Please add the following markup below our 'Send Message' <mobi:fieldsetRow> but above the closing </mobi:fieldsetGroup>:

```
<mobi:outputList rendered="#{user.hasJoined}" styleClass="chat">
        <mobi:fieldsetRow>
            <ui:repeat value="#{chatRoom.messages}" var="message">
                <mobi:outputListItem group="true">
                    <h:outputText value="#{message}" />
                    <br />
                </mobi:outputListItem>
            </ui:repeat>
        </mobi:fieldsetRow>
</mobi:outputList>
```

Add CSS

Instead of using inline CSS, we will create a custom css file and add our page specific styles. In the **web** directory of the project, create the folder structure resources/css and custom.css. In custom.css, add the following:

```
.chat {
        height: 450px;
        overflow: scroll;
}
```

We also need to add the CSS stylesheet reference to our page. Add the following to the **<head>** section of the page, below the <mobi:deviceResource> tag so that this stylesheet takes precedence:

```
<h:outputStylesheet library="css" name="custom.css" />
```

Add User.java Managed Bean

User.java will contain fields specific to each user and the methods to join the chat and send a message. These methods use an injected instance of our soon to be added ChatRoom object. The ChatRoom class will be our main controller class.

Notice that the User() constructor calls *PushRenderer.addCurrentSession(ChatRoom.GROUP);*. When the User class is instantiated, the user's current session will be added to a render group. For this tutorial we will have one render group but more complex use cases may require that we have multiple render groups for multiple chat rooms, a lobby, private messages, etc..

Please add the User.java class and ensure that you generate the getters/setters:

```
@ManagedBean
@ViewScoped
public class User implements Serializable{

	private String username;
	private String message;
	private boolean hasJoined;

	// injected instance of chatRoom managed bean
 @ManagedProperty(value = "#{chatRoom}")
	private ChatRoom chatRoom;

	public User() {
		hasJoined = false;
		PushRenderer.addCurrentSession(ChatRoom.GROUP);
	}

	public void join(ActionEvent event) {
		hasJoined = true;
		chatRoom.addUser(this);
	}

	public void sendMessage(ActionEvent event) {
		chatRoom.addMessage(this, message);
	}

	//TODO GENERATE GETTERS/SETTERS
}
```

Add ChatRoom.java Managed Bean

ChatRoom.java will contain the application's main functionality such as adding/removing a user from the chat and sending a chat message to members of the chat room. In addition to adding a user's session to a render group, here are the only other two required Cloud Push calls:

- PushRenderer.render(GROUP, new PushMessage("Missed Chat Message", message)); - A priority push, with option to do a cloud push.
- PushRenderer.render(GROUP); - A standard Ajax Push notification.

The difference between the two is that the first call contains the option to send a PushMessage which will use a common network protocol to send either an email or SMS. For this specific tutorial we will be sending a 'missed chat' notification via email. Please add the following class to the application:

```
@ManagedBean
@ApplicationScoped
public class ChatRoom implements Serializable{

   private Map users = new Hashtable();
   private List messages = new Vector();
```

```
    public static final String GROUP = "main";
    private String subject = "subject";
    private String message = "message";

    protected void addMessage(User user,String message) {
        messages.add(user.getUsername( )+ " says: " + message);

        // if user is unavailable, send chat message to their email
        PushRenderer.render(GROUP, new PushMessage("Missed Chat Message", message));
    }

    public void addUser(User user) {
        if (!users.containsKey(user.getUsername())) {
            users.put(user.getUsername(), user);
            messages.add(user.getUsername() + " joined");
            PushRenderer.render(GROUP);
        }
    }

    public void removeUser(User user) {
        Object removedUser = users.remove(user.getUsername());
        if (removedUser != null) {
            messages.add(user.getUsername() + " left");
        }
        PushRenderer.render(GROUP);
    }

    public List getMessages() {
        return messages;
    }
}
```

Add Email Configuration

When a chat room user is not actively using the application (UI not visible), the application will be unable to deliver new messages to the user. As such, missed chat messages will be delivered to the user via a Cloud Push PushMessage email alert. For this tutorial, we will use a gmail account to send to our alert email. Here is what is required:

1. Add the mail.jar to the **WEB-INF/lib** folder
2. Add the following to the WEB-INF/web.xml:

```
<context-param>
    <param-name>smtp.host</param-name>
    <param-value>smtp.gmail.com</param-value>
</context-param>
<context-param>
    <param-name>smtp.from</param-name>
    <param-value>your_email@gmail.com</param-value>
</context-param>
<context-param>
    <param-name>smtp.port</param-name>
    <param-value>465</param-value>
</context-param>
<context-param>
    <param-name>smtp.user</param-name>
    <param-value>your_email@gmail.com</param-value>
</context-param>
<context-param>
    <param-name>smtp.password</param-name>
```

```
    <param-value>your_password</param-value>
</context-param>
<context-param>
  <param-name>smtp.security</param-name>
  <param-value>SSL</param-value>
</context-param>
<context-param>
  <param-name>smtp.verify-server-certificate</param-name>
  <param-value>false</param-value>
</context-param>
```

Send Cloud Push Email Alert

When the UI is inactive, we will alert the user to new chat messages via email thanks to Cloud Push. There are two steps required:

1. Add the following javascript call to the <head> tag of the page:

```
<script type="text/javascript">
      function sendEmail() {
          // If alert cannot be delivered, send via email
    ice.push.parkInactivePushIds('mail:email@address.com');
      }
</script>
```

The email address could be dynamically configured like so:

```
ice.push.parkInactivePushIds('mail:' + document.getElementById('email').value);
```

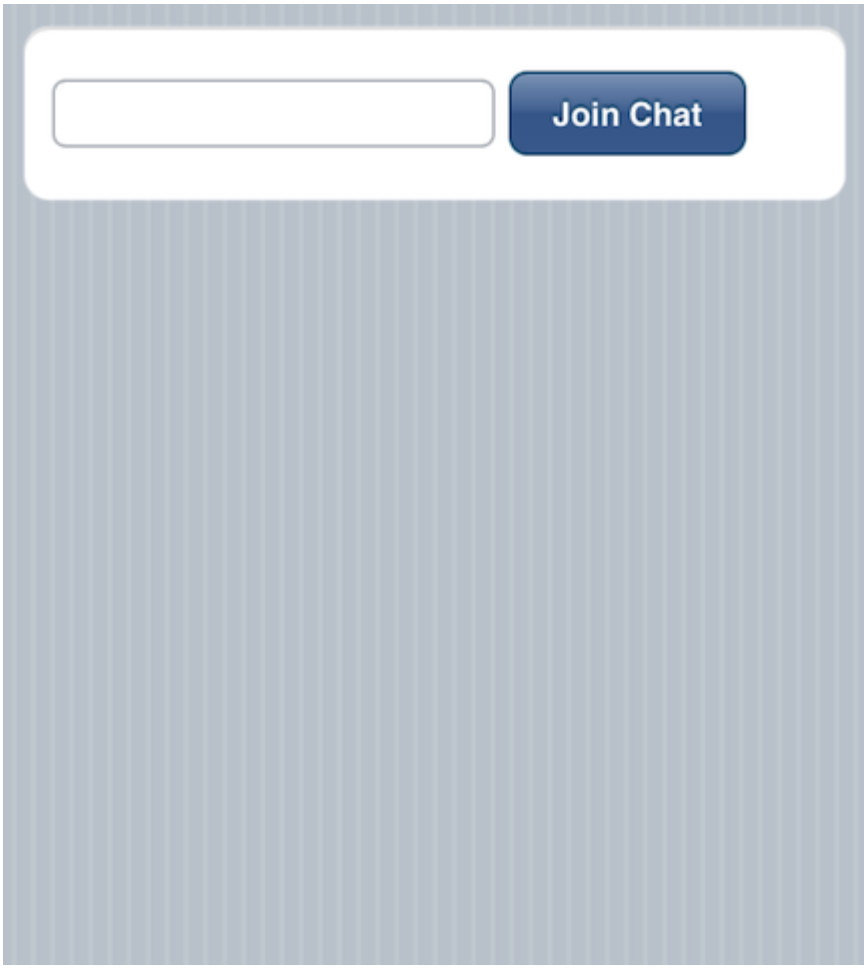
2. Add an onclick event to the 'Send Message' <mobi:commandButton>:

```
<mobi:commandButton onclick="sendEmail();" value="Send Message"
actionListener="#{user.sendMessage}" />
```

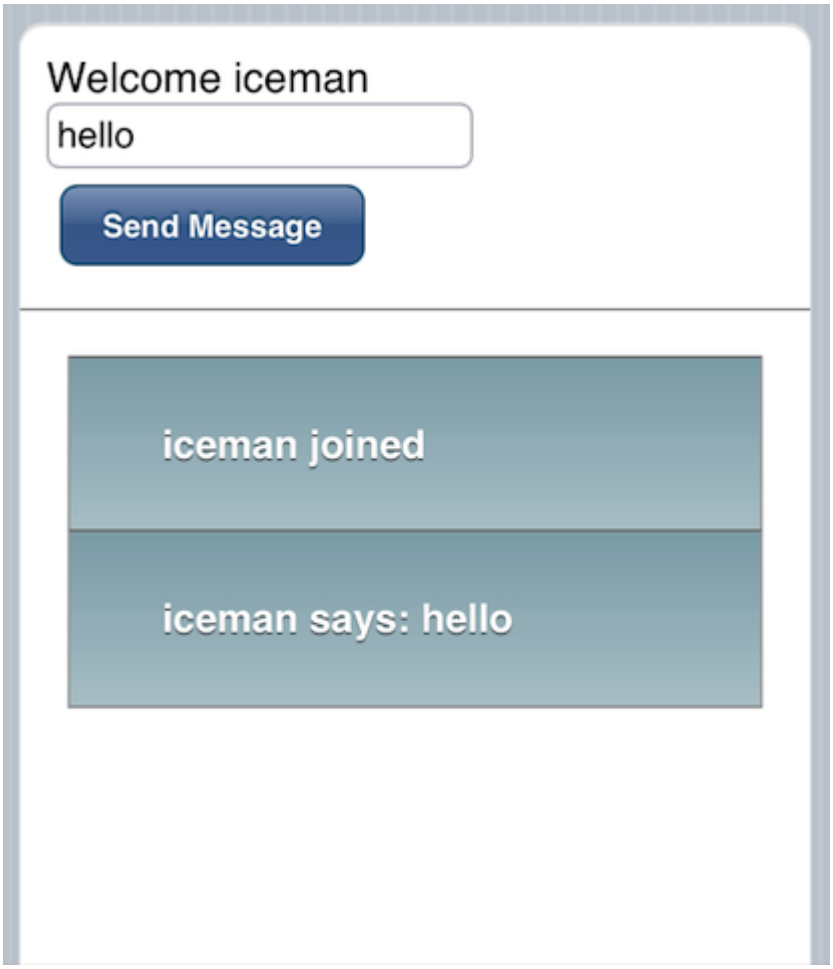
Run Application

Open two separate browsers (with 1 being a mobile browser) and login to the chat. Send a message and see that it is automatically delivered to all users of the chat room. If we minimize the application in our mobile browser, any missed messages to that device will be delivered to the user with the email configured in ice.push.parkInactivePushIds.

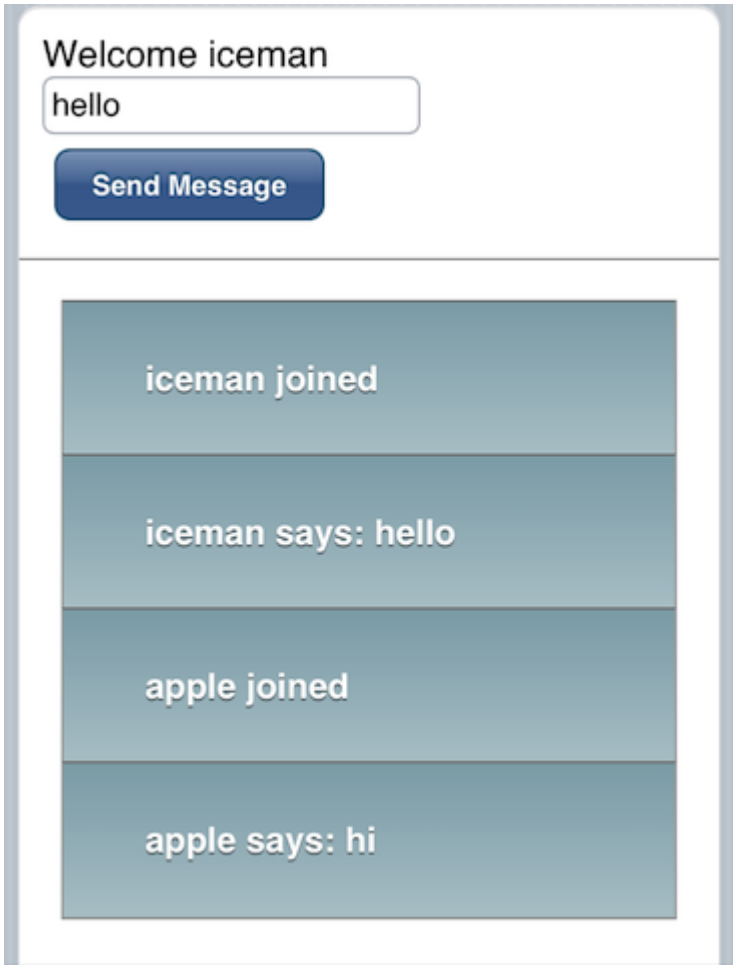
Enter a username to join the chat:



Send a message to all users of the chat:



Another user has sent a message:



Source

1. icemobile-cloudPush-tutorial.zip
2. mail.jar