ICEmobile : Cloud Push Overview

This page last changed on Feb 06, 2013 by steve.maryka.

Overview

User Notification

User Notification is the first aspect of cloud push that you must understand. When a user is actively working with an application on a mobile device, the user interface (UI) of that application can continually update information displayed to the user as the state of the application changes. For instances, an email application can update the number of emails in the inbox as new emails arrive. When the user is not actively using an application, that application can not rely on its UI to convey new information to the user, because the UI is not visible at the moment. Mobile devices provide a number of ways to alert the user of a state change occurring in an inactive application. A status bar can display an alerting icon, or a sound or vibration can occur. For instance, an email application can display an icon indicating that new mail has arrived since the user last viewed her inbox.

Cloud Push

When a user notification is the result of some state change coming from outside the device over a network connection, we refer to that as Cloud Push. The dominant mobile platforms all provide Cloud Push mechanisms that can be incorporated into an application. Specifically, the mechanisms are:

- Apple Push Notification Service
- Android Cloud To Device Messaging
- Blackberry Push Service

These mechanisms can be integrated directly with a native mobile application to achieve appropriate user notifications triggered from the network cloud. The [ICEmobile Containers] include the integration necessary to use device-specific cloud push mechanisms.

Beyond these proprietary push mechanisms, other common network protocols can be considered for cloud push notification, such as:

- Email
- SMS text messaging

These mechanisms have the advantage of ubiquitous support across mobile platforms, relying on the device's built in support for these protocols, but won't necessarily integrate tightly with the native application. ICEmobile currently supports a generic email connector for cloud push.

Web Applications

The concept of push in web applications was completely foreign until the advent of Ajax, and more specifically Ajax Push. Ajax Push makes pushing asynchronous state changes to a web application possible, so an active web application can behave much like an active native application, pushing state changes to the UI in a spontaneous fashion as the state of the application changes.

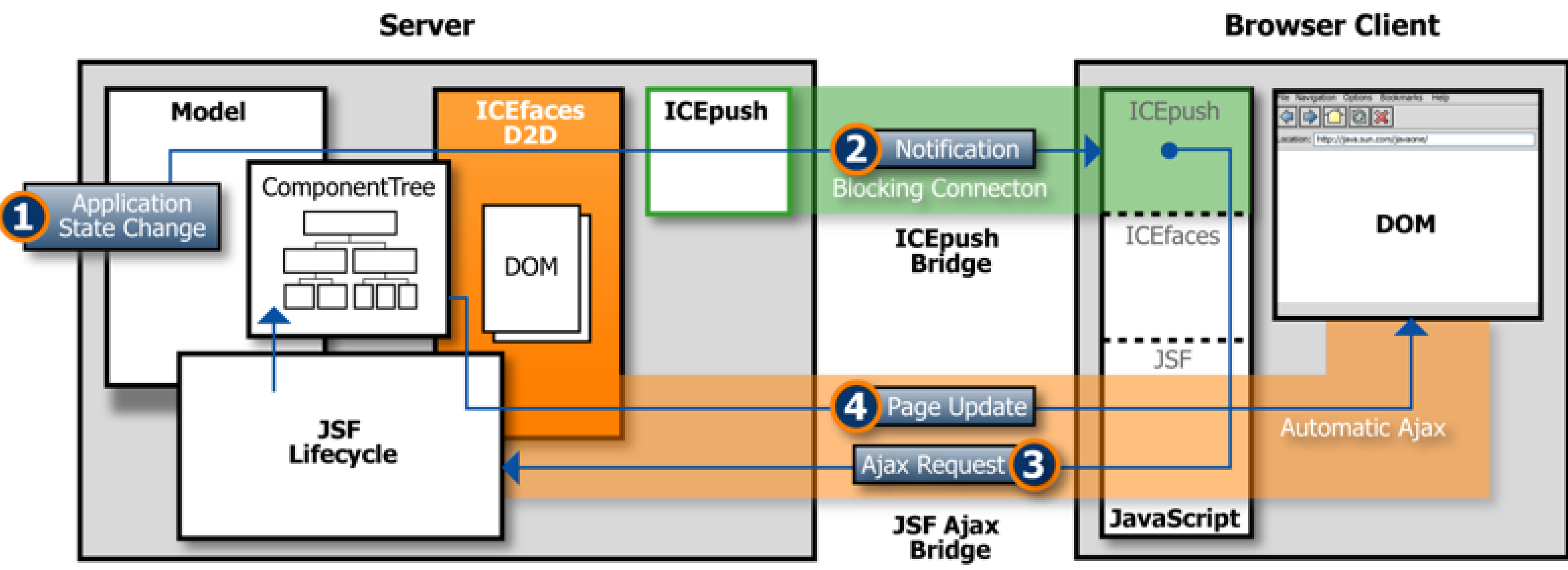
When a web application becomes inactive, not only is the UI unavailable, the network connection back to the server is also unavailable, so Ajax Push connections cannot be maintained. In order to support cloud push to web applications, ICEmobile-Faces augments Ajax Push with a priority push mechanism that utilizes a cloud push connector to a native [Device Container]. Even when the web application is idle, Ajax Push can send priority notifications over a cloud push connector to the device, and alert the user with standard user notification mechanisms on the device.

Ajax Push

We now review the Ajax Push mechanism with respect to active and inactive web applications.

Push to Active Applications

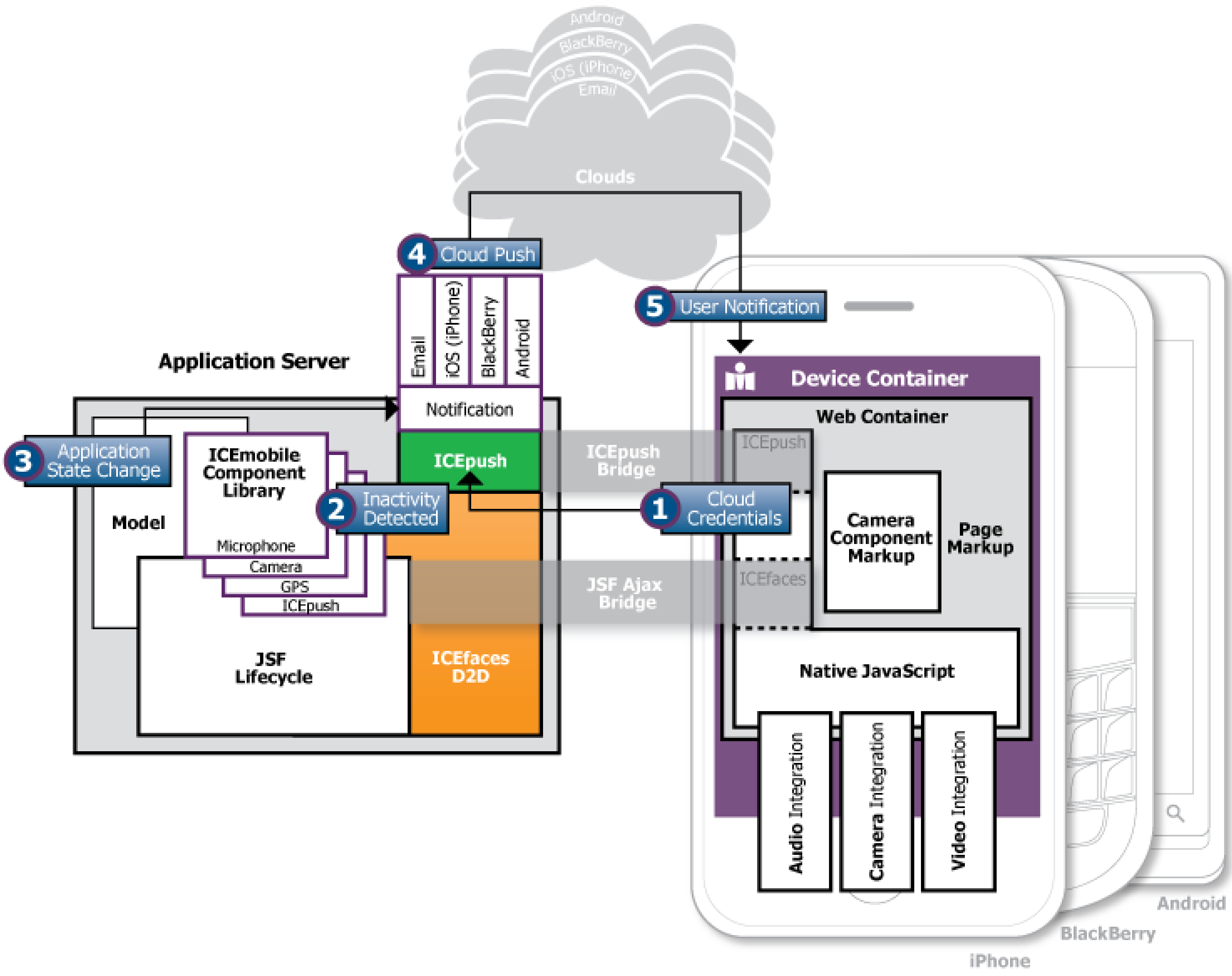
When the web application is active, the Ajax Push mechanism relies on HTTP and the notification protocol in ICEpush, which uses reverse polling over a blocking connection to facilitate asynchronous page updates. The basic process is illustrated below.



1. Some state change in the application triggers an Ajax Push event.
2. ICEpush notification is delivered to the browser client over a dedicated ICEpush connection.
3. Notification at client browser causes a JSF Ajax request with an empty execute phase.
4. Render phase captures new state of client, and Direct-to-DOM Rendering delivers incremental page updates to client.

Push to Inactive Applications

Once an ICEfaces page loads in the ICEmobile device container, the container notifies the ICEpush bridge, describing the cloud push connector type/credentials associated with the client device (if there is one). When the web application goes inactive on the device, the ICEpush bridge connection is parked on the server for that client's session. While the connection remains parked, any priority push requests that occur for that client will be sent via the specified cloud connector, to be received at the device container (in the case of a platform-specific connector), or by the email client (in the case of an email connector). In the case of the device container, the cloud push will result in a user notification carrying a specific subject/message. The process is illustrated below.



1. Cloud Push park credentials established.
2. Inactive state detected and client session parked.
3. Some state change in the application triggers a priority Ajax Push event.
4. ICEpush notification is delivered to the device using the specified cloud connector.
5. Notification received at device, and user notification is given.

At some time later the user acknowledges the notification by reactivating the application, causing the ICEpush bridge to reestablish itself, and a page update to occur reflecting the current state of the application.

Programming Model

While there is some relatively complex infrastructure in ICEmobile-Faces to handle cloud push, the programming model is simple, and well-aligned with the standard ICEfaces 2 Ajax Push APIs.

A standard Ajax Push notification is generated using:

```
import org.icefaces.application.PushRenderer;
...
```

```
PushRenderer.render("myGroup");
```

A priority push, with option to do a cloud push is generated using:

```
import org.icefaces.application.PushRenderer;
import org.icefaces.application.PushMessage;
...
PushMessage myMessage = new PushMessage("myMsgSubject", "myMsgBody");
PushRenderer.render("myGroup", myMessage);
```

From the developers perspective, there are no additional considerations beyond supplying the additional PushMessage. Selection of the normal or cloud push transport, as well as selection of the cloud push connector type, is transparent to the developer and handled by the ICEpush core.

Deployment Infrastructure

Specific deployment infrastructure is required on both the server and client for cloud push to operate correctly.

⚠ Device-specific cloud push features require ICEmobile-EE capabilities, and are not discussed here. Email is the only cloud push connector available in ICEmobile Open Source, and is the focus of the remainder of this tutorial.

Server Requirements

Deployments that support cloud push must include proper configuration for each cloud push mechanism to be supported.

Email Connector

The email connector requires a SMTP server account for sending the email messages. Your web application can be configured to login into the server with context parameters in the web.xml. The following attributes are available to configure the email client.

Attribute	Description	Value	Default
smtp.host	host name of SMTP server	any valid host name	none
smtp.from	email address of sender	any valid email address	none
smtp.port	port number for SMTP server	any valid port	25
smtp.user	user name for SMTP server access	any valid user name	none
smtp.password	password for user	user password	none
smtp.security	type of security for SMTP connection	NONE, SSL, TLS	NONE
smtp.verify-server-certificate	certificate verification	true, false	false

Device Requirements

Email

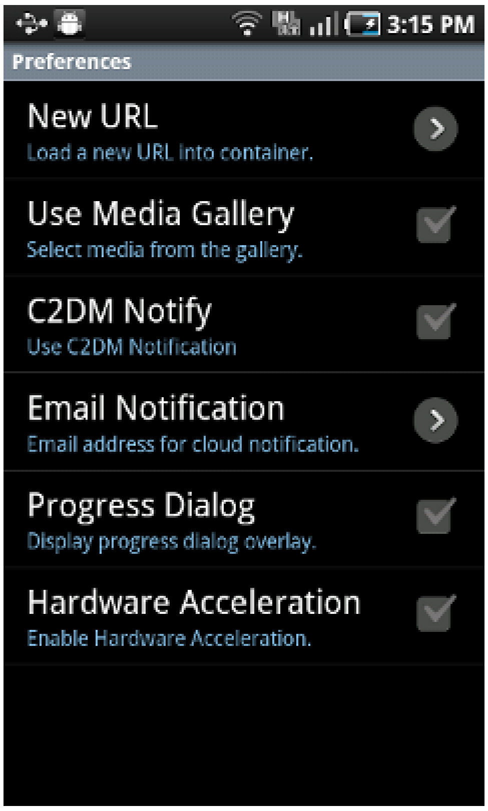
If the email connector is used, there must be an email client configured on the device to match the email address associated with the client. It is also necessary for the device to provide email credentials to the web application. This can be done via the [ICEmobile Containers], or via the web application itself.

⚠ See example below for details on how to supply credentials in non-container deployments.

.
Configuring the various device containers for email-based cloud push is now described.

Android

Cloud push credentials are established using the device container preferences page as illustrated below:



3:15 PM

Preferences

New URL
Load a new URL into container.

Use Media Gallery ✓
Select media from the gallery.

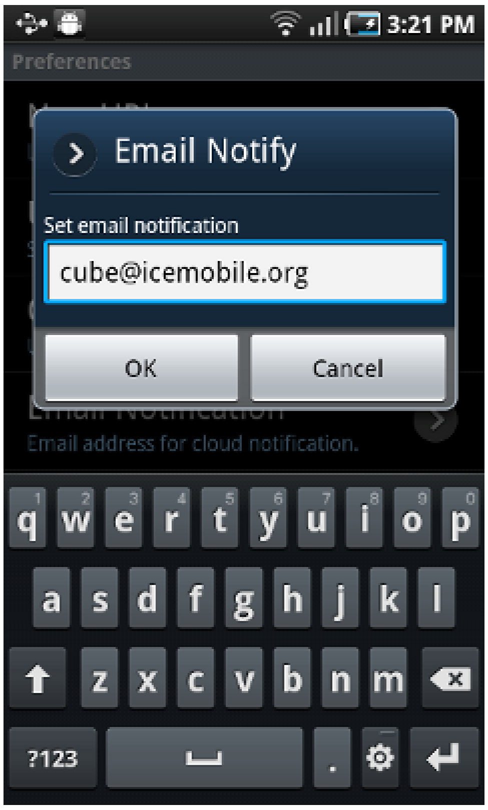
C2DM Notify ✓
Use C2DM Notification

Email Notification
Email address for cloud notification.

Progress Dialog ✓
Display progress dialog overlay.

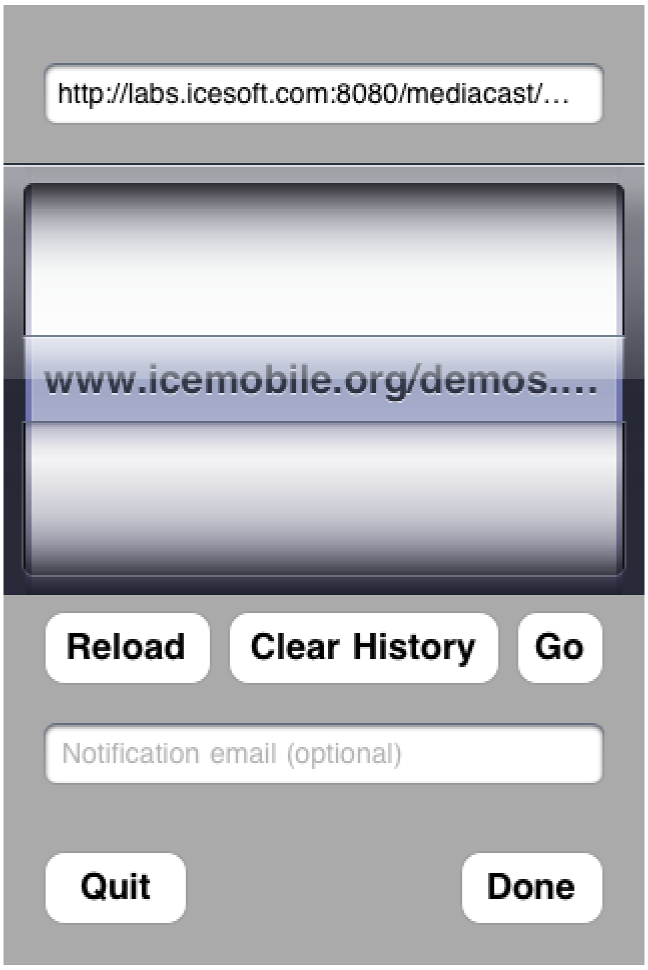
Hardware Acceleration ✓
Enable Hardware Acceleration.

To use email notification, C2DM must be deselected, as shown above, and a valid email address must be provided in the Email Notification preference as shown below.



iOS

To configure email notification in place of APNS, provide a notification email address in the preferences screen as seen below:

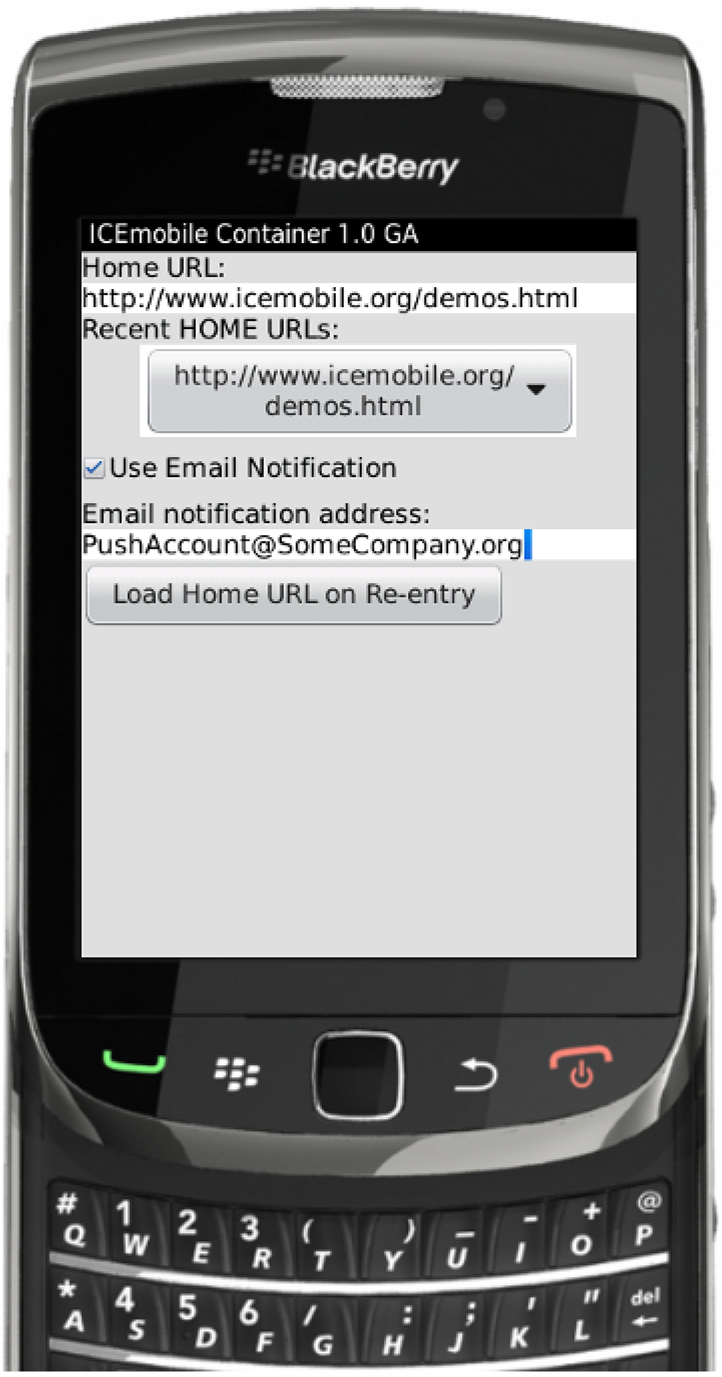

http://labs.icesoft.com:8080/mediacast/...
www.icemobile.org/demos....
Reload
Clear History
Go
Notification email (optional)
Quit
Done

Blackberry

To configure the Blackberry container for email notification, go to the options page for the ICEmobileContainer, under Options (wrench application) -> Third Party Applications -> ICEmobile Container 1.0 GA

Check the 'Use Email Notification' checkbox and the 'Email notification address' text field will become enabled. Enter an email address to receive email push notifications.

Save the settings on exit. The container will automatically re-register for push notifications with the new service.



ICEmobile Container 1.0 GA

Home URL:
http://www.icemobile.org/demos.html

Recent HOME URLs:
http://www.icemobile.org/demos.html

☑ Use Email Notification

Email notification address:
PushAccount@SomeCompany.org

Examples

Email Connector with Device Containers

This example provides deployment configuration for applications that use cloud push via the email connector, and device containers to supply necessary cloud push credentials from the client. We will use the mobileshowcase application supplied in the download bundle, and in particular the Notification page in mobileshowcase.

Deploying mobileshowcase

If you are using the binary bundle, then a prebuilt version of mobileshowcase.war is available in [install-dir]/samples/dist. If you are using the source bundle, from [install-dir] execute the "ant" command to build. When the build is complete, mobileshowcase.war will be available in [install-dir]/samples/dist. Deploy mobileshowcase.war to your application server, in the normal fashion and ensure that the applications is functioning.

Configure Email Connector

For Email Connector to function, you must supply valid SMTP server configuration and user credentials in the deployed web.xml file. In this example, we will illustrate the use of the gmail SMTP server, which you can use as well, provided that you have a valid gmail account.

Edit your web.xml to add the following configuration for SMTP.

```
<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>yourAccount@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>yourAccount@gmail.com</param-value>
</context-param>
<context-param>
     <param-name>smtp.password</param-name>
     <param-value>yourPassword</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>
```

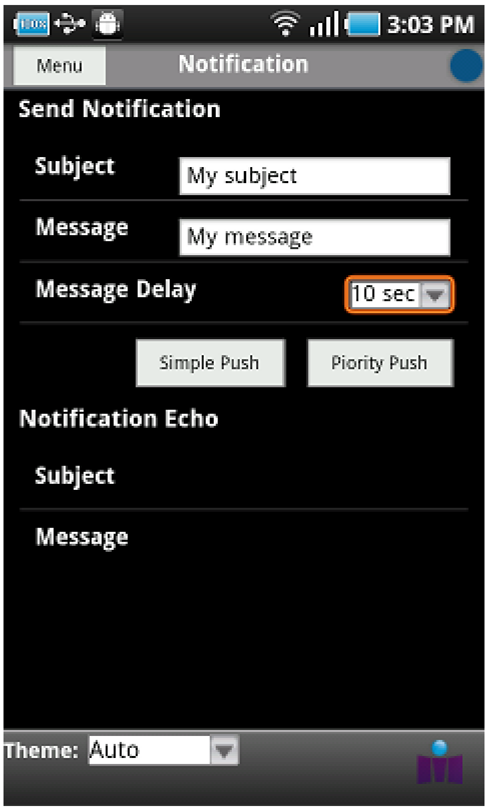
Substitute your gmail account and password as appropriate. Save the changes and restart the application. The server log from application startup should indicate that the Email Connector has been configured.

```
Jan 27, 2012 2:56:32 PM org.icepush.EmailNotificationProvider$AutoRegister contextInitialized
INFO: ICEpush Email Notification Provider Registered.
```

Testing your deployment

You will need a supported mobile device or simulator, and you will need the corresponding Device Container installed and configured. See above for information on how to configure your container to supply email credentials.

Once the container is configured, navigate, from the container, to the URL for your test deployment, and select the Navigation example. Here you can enter a subject and message, and set a delay for notification delivery. You can then select either Simple or Priority Push.



The following table illustrates the expected results under various conditions.

Client	Push Type	Result
Active	Simple	Normal push with page update
Active	Priority	Normal push with page update
Inactive	Simple	No notification or push
Inactive	Priority	Email notification

Email Connector without Device Containers

⚠ Still to come...