Introducing Presence and the Contact Roster

Presence is a lightweight mechanism used in instant messaging to broadcast a user's availability. Originally, presence was represented as a simple fl ag that indicated when a user was logged on and available to chat. This has gradually evolved into a more detailed status indicator that lets users describe their availability more accurately by indicating if they're available, busy, away from the computer, or offl ine. The recent popularity of applications like FriendFeed and Twitter has resulted in presence being expanded to include custom messages that can describe anything from a user's current activity to the music they're listening to.

Users can see the presence of all the people in their contact roster. The *contact roster* is a list of all the contacts with whom a user has an agreement to exchange messages and share presence information.

When adding someone to their roster, users are implicitly subscribing to updates of that person's presence, and changes to their own presence are propagated to all the contacts on their roster.

Instant messaging is an inherently portable technology — a user's presence and contact roster are maintained by the GTalk server, so the roster on an Android device is synchronized with Gmail chat and any desktop IM clients.

Managing the Contact Roster

Developers can access the contact roster to determine the presence of any of a user's IM contacts, monitor presence updates, add new contacts, remove existing ones, and handle subscription requests.

Accessing the IM Contact Roster

When it's made available, the contact roster should be accessible through a native Content Provider using the helper class android.provider.Im.Contacts. You can query it as you would any other Content Provider.

In the following snippet, you can see how to iterate over the roster to fi nd the presence of each IM contact:

```
Uri uri = android.provider.Im.Contacts.CONTENT_URI_CHAT_CONTACTS;
Cursor c = managedQuery(uri, null, null, null);
if (c.moveToFirst()) {
do {
String username = c.getString(c.getColumnIndexOrThrow(Contacts.USERNAME));
int presence = c.getInt(c.getColumnIndexOrThrow(Contacts.PRESENCE_STATUS));
if (presence == Contacts.AVAILABLE) {
// TODO: Do something
}
} while (c.moveToNext());
}
```

Monitoring the Roster for Changes

To monitor the roster for changes and presence updates, implement an IRosterListener and register it with an IM Session using addRemoteRosterListener, as shown in the skeleton code below:

```
IRosterListener listener = new IRosterListener.Stub() {
public void presenceChanged(String contact) throws RemoteException {
// TODO Update the presence icon for the user.
}
public void rosterChanged() throws RemoteException {
// TODO Update the roster UI.
}
public void selfPresenceChanged() throws RemoteException {
// TODO Update the user's presence.
}
};
try {
imSession.addRemoteRosterListener(listener);
} catch (RemoteException e) { }
```

The Roster Listener includes event handlers that will be triggered when a contact has been added or removed from the current user's roster, when a contact's presence has changed, and if the user's presence has changed.

Adding Contacts to a Roster

To add a new contact to the user's roster, use addContact, specifying the contact username and a personal nickname to customize their entry on the roster, as shown below:

```
imSession.addContact("jim@dundermifflin.com", "Big Tuna", null);
```

The specifi ed nickname is private and will only be visible to the device user.

People are only added to the roster after they've approved the request to become an instant messaging contact. After you attempt to add a contact, the target user receives an invitation (represented as a subscription request) that he or she can either approve or decline.

If the target user accepts the invitation, your user is placed in the target user's roster (and vice versa), and he or she will be able to exchange instant messages and receive presence updates. Subscription requests are asynchronous, so you'll need to listen for changes in the roster to determine when a subscription request has been granted.

Handling Subscription Requests

Requests from others to add the device user to their contact lists should be presented to the user for his or her explicit approval or rejection.

Once the user has indicated his or her preference, you can approve or decline subscription requests using the approveSubscriptionRequest and declineSubscriptionRequest methods on an IM Session. As shown below, both methods take a contact name as a parameter; the approve method also accepts an optional nickname for the new contact being added.

```
imSession.approveSubscriptionRequest(sender, "nickname", null);
imSession.declineSubscriptionRequest(sender);
```

Removing and Blocking Contacts

In these times of fl eeting attention and fi ckle friendships, there may come a time when a contact once added to a roster is no longer considered worthy of the honor. In extreme cases, users may choose to block all messages from a particular user.

Call removeContact from an IM Session to remove a contact from the user's roster and unsubscribe from his or her presence updates.

```
imSession.removeContact("whathaveyoudoneforme@lately.com");
```

When ignoring someone isn't enough, users can choose to block their messages entirely. The blockContact method effectively reverses the initial subscription-request approval and automatically denies any new subscription requests:

```
imSession.blockContact("ex@girlfriend.com");
```

Blocked contacts are added to the users "blocked list," which, like the roster itself, resides on the server. A contact blocked from Android will also be blocked in all other Google Talk clients.

Managing the User's Presence

The presence of the logged-in IM Session user is available using the getPresence method, as shown in the snippet below:

```
Presence p = imSession.getPresence();
```

This Presence object can be used to determine the user's IM visibility, his status, and any custom status message.

To change the user's presence, modify the Presence object and transmit it to the instant messaging server by calling setPresence on the IM Session.

The following code snippet shows how to set the user presence to DO_NOT_DISTURB and specifi es a custom status message:

```
String customMessage = "Developing applications for Android. Professionally";
p.setStatus(Presence.Show.DND, customMessage);
imSession.setPresence(p);
```

Changes to a user's presence won't take effect until after they've been committed on the server. The best practice is to use a Roster Listener to react to the change in the user's presence once it's been applied on the server side.