Sending and Querying Presence

After signing in to an XMPP server, your application should post the user's presence to the server, and also request presence information for the user's roster members (buddy list) on the server. The server will reply with the presence information (away, offline, and so on), and the full JID of each member currently signed on. You can use the retrieved JID and presence information to send a connection request to a user. The server will continue to send presence notifications until the connection is ended.

libjingle provides the following two helper **XmppTask** objects to send and request presence:

- **PresenceOutTask** sends your presence to the server. You must pass in a **Status** object populated with your status information.
- **PresencePushTask** is a listener that receives presence notifications from the server. You start it by calling **PresencePushTask::Start**. You must connect to this class's SignalStatusUpdate signal to be alerted when a presence stanza is received. This signal will be sent once for each roster member whose presence is received.

To run these (and any other **Task**) objects, you must instantiate them and pass in the name of the parent object (a parent object based on **Task**—usually **XmppClient**), populate them with any required information, and start them by calling their **Start** method.

> **Important: Task objects must be started by calling the Start method.** Although **Task** objects typically expose additional methods that you want to run (for example, **Send**), these methods typically only specify the desired action; the task does not actually start running until **Start** is called.

The following steps describe how to send and receive presence:

1. **Sign up to receive presence notifications.** Instantiate and start **PresencePushTask** and connect to its **OnStatusUpdate** signal. Some servers will return roster presence notifications immediately after receiving a presence stanza. To avoid missing this initial notification, you should create a listener before sending your own presence.
2. **Populate a status object with your own presence information.** The **Status** object holds the standard presence information, such as status, show, visible, and other information.
3. **Send your status to the server.** Instantiate and start **PresenceOutTask** to send your status to the server. The server should reply with presence notifications about all roster members after it receives your status.

After sending and receiving presence, you can hook up the session management pathway, the precursor to sending and receiving session requests.

The following code from callclient.cc demonstrates creating both these objects to send and receive presence information. It is trimmed and reorganized for clarity.

```
// Create a PresencePushTask object to retrieve presence information from the server.
// Create this first to avoid missing a notification received by the sign in process.
presence_push_ = new buzz::PresencePushTask(xmpp_client_);

// Hook up to the notification signal sent when the stanza is received.
presence_push_->SignalStatusUpdate.connect(this, &OnStatusUpdate);

// Start listening for presence stanzas.
```

```
presence_push_->Start();

// Populate an object to report your current status.
buzz::Status my_status;
my_status.set_jid(xmpp_client_->jid());
my_status.set_available(true);
my_status.set_invisible(false);
my_status.set_show(buzz::Status::SHOW_ONLINE);
my_status.set_priority(0);
my_status.set_know_capabilities(true);
my_status.set_phone_capability(true);
my_status.set_is_google_client(true);
my_status.set_version("1.0.0.66");

// Create the PresenceOutTask object to send your presence.
buzz::PresenceOutTask* presence_out_ = new buzz::PresenceOutTask(xmpp_client_);

// Send your presence to the server. This is two steps:
// 1) Send() tells it what you want it to do.
// 2) Start() actually starts the task (sends the information and
//    waits for the proper response.
presence_out_->Send(my_status);
presence_out_->Start();

// Called by PresencePushTask::SignalStatusUpdate whenever a status
// notification is received.
void OnStatusUpdate(const buzz::Status& status) {
  RosterItem item;
  item.jid = status.jid();
  item.show = status.show();
  item.status = status.status();
  std::string key = item.jid.Str();

  if (status.available() && status.phone_capability()) {
     cout << "%s is available and ready to chat." << key.c_str();
  } else {
    cout << "%s is no longer able to chat." << key.c_str();
  }
}
```


Last updated March 23, 2012.