# Making and Receiving Connections

Once you have signed up and posted your presence, and received presence notifications from the XMPP server, you can start to make connection requests (or answer incoming requests). It is important to note that when someone sends you a connection request, you will get an asynchronous notification of this request, but your computer will begin negotiating connections immediately, without waiting for your response. Depending on the libjingle code used, data exchange may also begin immediately. For example, in the File Share example, the caller starts sending the image files immediately as soon as the connection has been negotiated (which may be before the recipient has accepted the connection request), although file transfer will not begin until the other client has explicitly accepted the connection. Likewise, in the voice chat example, sound bytes can begin flowing as soon as the connection is made.

**Sending a connection request**

1. **Send connection requests.** Sending a connection request means creating a new **Session** object, creating a description of session-specific information (codecs, file names, and so on), and sending them to the other computer, and then monitoring **Session::SignalState** for in formation about the request. The details of connecting are specific to the session type: see the voice chat application and file share application for details. Under the covers, **Session::Initiate** is called with the JID of the recipient, and a **SessionDescription** subclass describing the details of that session (files being sent or requested, codecs available, etc).
2. **Respond to incoming connection requests.** When a new session request is received, a **Session** object will be created for you automatically, and **SessionClient::OnSessionCreate** will be called, with a flag indicating whether this is an incoming or outgoing request. For incoming requests, you should notify the user, who can accept or reject the request. The user must call **Session::Accept** or **Session:Reject** to accept or reject the request. The details of this are specific to the session type: see Voice Chat Application or File Share Application for details.
3. **Send and receive data.** Data is sent or received through the **TransportChannel** object, but how this is managed varies widely by session type. For voice calls, **MediaEngine** calls **SendPacket**, and **TransportChannel** calls **ReadPacket** to send and receive data. However, if you create your own application using the **StreamInterface** wrappers for local data sources (such as **MemoryStream**), you will have to call read and write methods yourself. See **StreamInterface** for more information.

The following code from callclient.cc demonstrates the MakeCallTo method, which is called with the bare JID of a caller to contact. It also demonstrates how CallClient is notified for an incoming call.

```
// Call this function with name = <bare JID of a caller to contact>.
void CallClient::MakeCallTo(const std::string& name) {
        bool found = false;
        buzz::Jid found_jid;
        buzz::Jid callto_jid = buzz::Jid(name);

        // Iterate through the stored roster received from the service to find
        // a user with a matching bare JID.
        RosterMap::iterator iter = roster_->begin();
        while (iter != roster_->end()) {
                if (iter->second.jid.BareEquals(callto_jid)) {
                        found = true;
                        found_jid = iter->second.jid;
                        break;
                }
                ++iter;
        }
        if (found) {
                console_->Printf("Found online friend '%s'", found_jid.Str().c_str());

                // Connect to receive notification when a call is disconnected, just to
                // alert the user.
                phone_client()->SignalCallDestroy.connect(
                this, &CallClient::OnCallDestroy);

                // Create a new Call object and tell it to create a new Session object.
                // When the Session is created, Call will send OnSessionState, where we hook
                // into the important notifications. This is described below.
                if (!call_) {
                        call_ = phone_client()->CreateCall();
                        console_->SetPrompt(found_jid.Str().c_str());
                        call_->SignalSessionState.connect(this, &CallClient::OnSessionState);
                        session_ = call_->InitiateSession(found_jid, NULL);
                }
        phone_client()->SetFocus(call_);
        } else {
        console_->Printf("Could not find online friend '%s'", name.c_str());
        }
}

...
// Notified when a Session object changes state.
// If this is an incoming call, we'll get STATE_RECEIVEDINITIATE.
// If this is an incoming call, we are able to receive this notification because
// of the following lines of code elsewhere in the file:
//              Called when PhoneSessionClient is created:
//              in the initialization method for CallClient,
//   phone_client_->SignalCallCreate.connect(this, &CallClient::OnCallCreate);
//   ...
//
//void CallClient::OnCallCreate(cricket::Call* call) {
//  call->SignalSessionState.connect(this, &CallClient::OnSessionState);
//}
//
void CallClient::OnSessionState(cricket::Call* call,
                                cricket::Session* session,
                                cricket::Session::State state) {
  if (state == cricket::Session::STATE_RECEIVEDINITIATE) {
    buzz::Jid jid(session->remote_name());
    console_->Printf("Incoming call from '%s'", jid.Str().c_str());
    call_ = call;
        session_ = session;
        incoming_call_ = true;
        // Client should call Session::Accept to accept the connection.

  } else if (state == cricket::Session::STATE_SENTINITIATE) {
    console_->Print("calling...");
  } else if (state == cricket::Session::STATE_RECEIVEDACCEPT) {
    console_->Print("call answered");
  } else if (state == cricket::Session::STATE_RECEIVEDREJECT) {
    console_->Print("call not answered");
  } else if (state == cricket::Session::STATE_INPROGRESS) {
    console_->Print("call in progress");
  } else if (state == cricket::Session::STATE_RECEIVEDTERMINATE) {
    console_->Print("other side hung up");
  }
}
```

*Last updated March 23, 2012.*