SessionManager Class

SessionManager creates, destroys, and routes XMPP to and from individual **Session** objects. You do not need to extend this object for different session types.

The application must create this object and connect to **SignalRequestSignaling**, and in response to that signal, call **OnSignalingReady**. The methods shown are not thread-safe, and can be called on any thread, but you should call them from the signaling thread.

Syntax

```
class SessionManager : public sigslot::has_slots<>
```

Methods

Name	Description
~SessionManager	Destructor. Does not deallocate any **Session** objects, but does empty its local list of pointers.
bool **IsSessionMessage**(const buzz::XmlElement* *stanza*)	Determines whether a stanza should be forwarded to a **Session** object.
int **session_timeout**()	Used by **Session** to determine how long it can be unwritable before it should send a timeout error and terminate itself. **Session** multiplies this by 1,000 so it can be considered a time in seconds.
PortAllocator* **port_allocator**()	Returns the **PortAllocator** subclass used by this object.
Session ***GetSession**(const SessionID& *id*);	Returns the session with the given ID or NULL if none exists
Session* **CreateSession**(const std::string &*name*, const std::string &*session_type*)	Called to create a session. • *name* A unique string used by libjingle to identify the session. libjingle uses the JID of the other party. This is used to identify the session in XMPP stanzas, and enables **SessionManager** route the message to the proper Session object. • *session_type* An ID to identify the **SessionClient** associated with this Session. See **AddSession**.
Session* **FindSessionForStanza**(const buzz::XmlElement* *stanza*, bool *incoming*)	Finds a Session for an incoming XMPP stanza. The *name* parameter passed in to CreateSession is used to find the correct **Session**.
Session* **GetSession**(const SessionID &*id*)	Returns a session by **SessionID** (a JID plus a random string).

SessionClient* **GetClient**(const std::string& *session_type*)	Returns the **SessionClient** subclass responsible for a **Session**, identified by a string ID.
SessionManager(PortAllocator **allocator*, Thread **worker_thread*=0)	Constructor. • *allocator* The PortAllocator subclass to use. This is instantiated by the application. • *worker_thread* An optional thread used to handle peer-to-peer tasks such as port allocation. If not specified, this will be the same as the thread the object is built on.
talk_base::Thread **worker_thread**()	Returns the worker thread passed in to the constructor, or the **SessionManager** main thread if no thread was submitted. This thread is used by the socket manager to handle peer-to-peer socket processing.
Thread* **signaling_thread**()	Returns the thread that this object was instantiated in.
void **AddClient**(const std::string& *session_type*, SessionClient* client)	Registers a **SessionClient** subclass responsible for a particular **Session** type. Each Session has an associated string type ID, passed through XMPP initiation requests. libjingle registers various session types as global variables. New types should be registered on both the sender and receiver, You should register your handler for each type of **SessionClient** subclass instantiated. When a session request is received from another computer, SessionManager will create a new Session object and then call that object's **SessionClient::OnSessionCreate** method.
void **DestroySession**(Session **session*)	Sends a **SignalSessionDestroy** signal (which is caught by **SessionClient**) and removes the client from its internal list, but does not destroy the object itself.
void **OnIncomingMessage**(const buzz::XmlElement* I)	Called when we receive a stanza for which **IsSessionMessage** is true
void **OnIncomingResponse**(const buzz::XmlElement* *orig_stanza*, const buzz::XmlElement* *response_stanza*);	Called when we get a response to a message that we sent.
void **RemoveClient**(const std::string& *session_type*)	Removes a **SessionClient** from the ID/**SessionClient** map.
void **set_session_timeout**(int *timeout*)	Sets the session timeout. See **session_timeout**.
void **TerminateAll**()	Calls **Session::Terminate** on all sessions. Currently not used.

Signals

SignalSessionCreate< Session *, bool >
Sent when **SessionManager** creates a new **Session** object (in **CreateSession**).

SignalSessionDestroy< Session * >
Called by **DestroySession**.

SignalOutgoingMessage< const buzz::XmlElement* >
Called to send an outgoing message. **SessionManagerTask** subscribes to this to send XMPP messages.

SignalRequestSignaling
Called by **OnRequestSignaling** to verify that the signaling channel is open. The application should connect to this signal and reply by calling **SessionManager::OnSignalingReady**.

SignalSendResponse< Session *, const SessionResponseMessage * >

Sends an error message stanza in reply to an incoming stanza that could not be routed properly or understood. This is caught by **SessionClient** which sends it out.

Attributes: public

Declaration file: talk/p2p/base/sessionmanager.h