Session Class

Negotiates the specifics of the data channel over the XMPP channel, monitors the connection, starts, and ends the connection. The **Session** object is created by the **SessionManager** object when a new connection request is made, whether or not the connection has been accepted by the application. When this object is created, it immediately calls **SessionClient::OnSessionCreate**; that method should connect to the Session object's SignalState signal and alert the user about the incoming connection request, to enable them to accept or reject the request by calling **Accept** or **Reject**.

When an incoming connection request is made, this class will be instantiated by the **SessionManager** when it receives an initiate stanza for a session that does not exist. For outgoing requests they are created explicitly at the request of another object, for example the **PhoneSessionClient**.

Each **Session** object has a unique ID that is added to XMPP stanzas that it sends or receives. **SessionManager** uses this ID to determine whom the stanza is for.

You should not need to modify or override this class. However, if you are creating your own custom sessions, you should know how to use this class from the logic component. 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 Session : public MessageHandler,
                public sigslot::has_slots<>
```

Methods

Name	Description
bool **Accept**(const SessionDescription **description*)	Accepts an incoming session request.
const SessionDescription* **description**()	Returns the **SessionDescription** object for this session.
Session::Error **error**()	Returns the last error value sent by SignalError.
const SessionID& **id**()	Returns the ID used to identify the **Session** in stanzas.
bool **Initiate**(const std::string &to, vector<XmlElement*> *extra_xml, const cricket::SessionDescription **description*)	Starts a request for a connection with another computer. The call initiates local candidate creation and will generate the full request and send it out through **SendSessionMessage**.
bool **initiator**()	Boolean value indicating whether this session was initiated by this computer, or by the other computer.
std::string& **name**()	Arbitrary string name for this session.
bool **Redirect**(const std::string	Called in response to an initiate or modify, to redirect the connection request. *target*

&*target*)	is the JID of the new target.
bool **Reject**()	Called to reject a connection request.
const SessionDescription* **remote_description**()	Returns the **SessionDescription** object sent by the other computer to describe a particular session request. This description is a custom set of information appropriate to the session type. For example, voice chat connections would include a codec list, and file transfers might include a file name.
std::string &**remote_name**()	Returns the JID of the other party in this session.
void **SendInfoMessage**(const Session::XmlElements &*elems*)	Sends arbitrary XML messages.
Session(SessionManager **session_manager*, const std::string &*name*, const SessionID &*id*)	Constructor. Creates a new session and SocketManager object, but does not actually start the connection process.
~Session	Destructor. Clears the SocketManager and removes the object from the signaling thread.
SessionManager* **session_manager**()	Retrieves the SessionManager object that manages this object.
bool **Terminate**()	Ends the session connection.
Transport* **transport**()	Returns the transport that was negotiated, or NULL if the negotiation is still in progress.
const std::string& **session_type**() const	A unique value that is used as a key by **SessionManager** to identify the **SessionClient** that handles this particular **Session** object. **SessionManager** keeps a map of sesson type/**SessionClient** instance values.
TransportChannel* **CreateChannel**(const std::string& *name*)	Creates a new channel with the given name. This method may be called immediately after creating the session. However, the actual implementation may not be fixed until transport negotiation completes. The name is a unique ID used to identify a specific channel in the Transport.
State **state**() const	Returns the state of the session. When this changes, the object sends SignalState.
TransportChannel* **CreateChannel**(const std::string& *name*	Creates a new channel with the given name. This method may be called immediately after creating the session. However, the actual implementation may not be fixed until transport negotiation completes.
Transport* **GetTransport**(const std::string& *name*)	Once transports have been created (by SetTransports), this function will return the transport with the given name or NULL if none was created. Once a particular transport has been chosen, only that transport will be returned.
void **SetPotentialTransports**(const std::string *names*[], size_t *length*)	Controls the set of transports that will be allowed for this session. If we are initiating, then this list will be used to construct the transports that we will offer to the other side. In that case, the order of the transport names indicates our preference (first has highest preference).

	If we are receiving, then this list indicates the set of transports that we will allow. We will choose the first transport in the offered list 1) whose name appears in the given list, and 2) that can accept the offer provided (which may include parameters particular to the transport). If this function is not called (or if it is called with a NULL array), then we will use a default set of transports.

Signals

SignalChannelGone<Session*, const std::string&>
Sent when we cannot find a matching channel on the other client.
SignalError<Session *, Error>
Sent when an error occurs in the session. The error is not necessarily a fatal error.
SignalInfoMessage<Session *, const XmlElements&>
Sent when an arbitrary XML message is received from the other client.
SignalState<Session *, State>
Sent when the state of the session changes, for example if the session changes from 'initiated' to accepted, to 'in progress', to 'ended.' The **SessionClient** implementation should connect to this signal in its **SessionClient::OnSessionCreate** method. If you connect at that time, you should receive all session messages, including STATE_RECEIVEDINITIATE for incoming messages.
SignalReceivedTerminateReason<Session *, const std::string &>
Sent when a termination message is received. The second parameter, if specified, can be a string description of why the session is being terminated.

Error

Session defines the following enumerations:

Value	Description
ERROR_NONE	No error.
ERROR_TIME	No response to a request to start signaling (**SignalRequestSignaling**).
ERROR_RESPONSE	An error occurred on the signaling thread.
ERROR_NETWORK	Network resources could not be allocated.

State

The **State** enumeration describes a **Session** state sent by **Session::SignalState** and **Call::SignalSessionState**.

Value	Description
STATE_INIT	This is a new session; no actions have yet occurred.
STATE_SENTINITIATE	Sent an initiation request; waiting for the other computer to accept or reject.
STATE_RECEIVEDINITIATE	Received a connection request from another computer; the application must call **Session::Accept** or **Session::Reject**.
STATE_SENTACCEPT	Sent an acceptance to a connection request.
STATE_RECEIVEDACCEPT	The other computer accepted your connection request.

STATE_SENTMODIFY	Sent a request to modify the connection in some way (for example, to change codecs).
STATE_RECEIVEDMODIFY	Received a request to modify the connection in some way (for example, to change codecs).
STATE_SENTREJECT	Sent a rejection to a connection request.
STATE_RECEIVEDREJECT	Your connection request was rejected.
STATE_SENTREDIRECT	Sent a request to redirect a connection to a new location.
STATE_SENTTERMINATE	Sent a termination notification to the other computer.
STATE_RECEIVEDTERMINATE	The other computer has ended the connection.
STATE_INPROGRESS	The connection has been made. This is not a guarantee that the connection is currently writable, but that both computers have negotiated a connection. If you are using the TCP example code, you must still either call **StreamInterface::GetState** or wait for **StreamInterface::SignalEvent** to return the SE_READ or SE_WRITE value as required.
STATE_DENIT	The session is being destroyed.

Attributes: public

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