Port Class

The **Port** base class wraps a local socket used to send and receive data. It monitors the connection health and read/write status, and has one or more **Connection** objects representing addresses on the remote computer. It is created by the **PortAllocator** subclass, and is managed by **Transport**. The methods shown are not thread-safe, and can be called on any thread, but you should call them from the worker thread.

Port is intended to be extended. libjingle includes the following subclasses of **Port**.

- **RelayPort** Handles a port on a relay server.
- **TCPPort** Handles a local TCP port. This is usable as it is, or it can be extended.
- **UDPPort** Handles a local UDP port. This is usable as it is, or it can be extended.

The port allocator assigns each port a preference, which is used to calculate which port should be used in a connection. This is a value from zero to one, where a higher number indicates greater preference. **BasicPortAllocator** preference values are prefixed with PREF_, and are defined at the top of basicportallocator.cc.

After creating the object, you must call **Start** to start the timeout monitor.

Syntax

```
class Port: public MessageHandler,
            public sigslot::has_slots<>
```

Enumerations

Name	Description
CandidateOrigin	Describes the direction of data flow in a connection. The following values are defined: • ORIGIN_THIS_PORT • ORIGIN_OTHER_PORT • ORIGIN_MESSAGE
Lifetime	Used to monitor the current status of the port in its march toward timing itself out. These are used internally, and need not
ProtocolType	Defines the protocol to be used to send data. The following values are defined: • **PROTO_UDP** UDP format. • **PROTO_TCP** TCP format • **PROTO_SSLTCP** TCP over SSL. • **PROTO_LAST** = PROTO_SSLTCP A stop value to indicate the number of available values.

Methods

Name	Description
void **AddAddress**(const SocketAddress &*address*, const std::string &*protocol*, bool *final*=true)	Adds the address of the local port.
void **AddConnection**(Connection **conn*)	Adds the given connection to the list of connections managed by this object.
const std::vector<Candidate>& **candidates**() const	Returns the address of this port.
const AddressMap& **connections**()	Returns a map of **Connection** objects linked to this port, keyed by remote address.
virtual Connection* **CreateConnection**(const Candidate &*remote_candidate*, CandidateOrigin *origin*)=0	Creates a new **Connection** object to the specified port. This does not add it to the list of **Connection** objects linked to this port; to do that, you must call **AddConnection**. • *remote_candidate* The remote candidate. • *origin* The direction that data will be flowing. See the enumerations table for a description of available values.
AsyncPacketSocket * **CreatePacketSocket**(ProtocolType *proto*)	Creates the underlying socket used by this port to send data. This is called by a UDP port when it is instantiated, or by other port types when it is needed.
uint32 **generation**()	The port allocator generates a group of ports upon request. Each time it does this, it assigns an incremented value to the generation attrubute of the port. **Transport** tries to prevent creation of multiple identical port types by removing identical port types created previously. This method returns the value assigned when the port was created. There is no default value; it must be set explicitly using **set_generation**.
Connection* **GetConnection**(const SocketAddress &*remote_addr*)	Returns a **Connection** object assigned to the specified remote address, if any.
virtual int **GetError**() = 0	Returns the error value last sent by the socket wrapped by this port. Error values are defined in talk/base/socket.h
bool **GetStunMessage**(const char **data*, size_t *size*, const SocketAddress &*addr*, StunMessage *&*msg*, std::string &*remote_username*)	Called by the **Connection** object to translate an incoming packet from a STUN packet into a simple data blob, and takes any STUN-specific actions required by the message (such as responding to a bind request). If this is not a STUN message, it will not modify the message. The method returns True if it was a STUN message, False if it was not. • *data* [in] The incoming data. • *size* [in] Size of the data, in bytes. • *addr* [in] The remote candidate address. • *msg* [out] The original packet if this was a stun message, the translated packet otherwise. This parameter is allocated by **Port**; the caller must release it. • *remote_username* [out] The remote fragment of the STUN user name.
std::string& **name**() const	A name specified by **set_name**. This name is only used for debugging

	purposes. There is no default value.
Network* **network**()	The network that this port was allocated on.
void **OnMessage**(Message **pmsg*)	Handles messages sent by other threads.
void **OnReadPacket**(const char **data*, size_t *size*, const SocketAddress &*addr*)	Called by the **Port** subclass when a packet is received from an unknown address (not in a **Connection** object managed by this object). If this were from a known address, it would have been forwarded to the **Connection** object that handled that address, which would have sent it out through **SignalReadPacket** to **P2PTransportChannel**.
std::string& **password**() const	A random password generated at construction time. This is to provide greater assurance that a packet is intended for the specific recipient; it is not to be considered a security measure. This value can be changed by calling **set_password**. A random default value is assigned on initialization. The password must be correct for the name returned by **username_fragment**.
Port(Thread **thread*, const std::string &*type*, SocketFactory **factory*, Network **network*)	Constructor. Autogenerates a name and password to use for this port. • *thread* The messaging thread for this object. • *type* The type of port to create. Each port defines a string that describes the port. This string is used by **BasicPortAllocator** to help determine a writable state. • *factory* An optional socket factory object used to create local sockets. If this is NULL, Port will create one by calling **Thread::socketserver** on *thread*.
~Port	Destructor. Deletes the references to all **Connection** objects.
float **preference**() cost	A preference for this port type. This is a value from 0 to 1, inclusive, where the higher number indicates greater preference. The preference for different port types is defined by the port allocator subclass. This value is set using **set_preference**.; the default value is -1.
virtual void **PrepareAddress**() = 0	Asynchronously tries to create a local address that the other computer can send to. When the address is ready, it calls **SignalAddressReady**. This is called by the port allocator when it creates a new port.
static const ProxyInfo& **proxy**()	Returns informatin about the proxy, if any, used by this port. This static value for all ports value is set by **set_proxy**.
void **SendBindingErrorResponse**(StunMessage **request*, const SocketAddress &*addr*, int *error_code*, const std::string &*reason*)	Sends an error response to a bind request from an unknown address. See **SignalUnknownAddress**.
void **SendBindingRequest**(Connection **conn*)	Creates and sends a STUN bind request for the submitted **Connection**.
void **SendBindingResponse**(StunMessage **request*, const SocketAddress &*addr*)	Sends a bind accept to a bind request.
virtual int **SendTo**(const void **data*, size_t *size*, const SocketAddress &*addr*, bool *payload*)=0	Sends a STUN packet to the address *addr* if a matching **Connection** object exists. If no such **Connection** object exists, this call will fail. This is called by a **Connection** subclass.

	• *data* The data to send • *size* Size of the data, in bytes. • *addr* Address of an existing **Connection** object handled by this port. • *payload* Whether or it is a simple payload packet. False if it is a STUN packet, True otherwise.
void **set_generation**(uint32 *generation*)	Sets the creation generation value for this port. See **generation**.
std::string& **set_name**(const std::string &*name*)	A name used to identify this port for debugging purposes. See **name**.
void **set_password**(const std::string &*password*)	Sets the password used to ensure proper port address. See **password**.
void **set_preference**(float *preference*)	Sets the preference for this port. See **preference**.
void **set_proxy**(const std::string &*user_agent*, const ProxyInfo &*proxy*)	Sets the proxy to use when connecting. It is called by **BasicPortAllocator** before it starts allocating ports. There are no default values if this method is not called. See **proxy**.
void **set_socket_factory**(SocketFactory **factory*)	Specifies the object that will create the local sockets. See **socket_factory**.
void **set_username_fragment**(const std::string &username_fragment)	Specifies the name that matches the object's password. See **username_fragment**.
int **SetOption**(Socket::Option opt, int value)=0	Enables the port creator to specify various port options. These options are implemented subclasses. Currently the only supported value is OPT_DONTFRAGMENT, with a value of 0 or 1, where 0 means that data can be split across packets, and 1 means that it cannot. **TunnelSessionClient** sets this option to 1. Returns zero if successful, nonzero otherwise.
SocketFactory* **socket_factory**() const	Retrieves object that will create the local sockets. This value is initially specified in the constructor, but can be changed by calling **set_socket_factory**.
void **Start**()	Starts the timer used to determine whether the port is necessary. If a port remains unused (without connections) for more than a specified period of time (defined by kPortTimeoutDelay in port.cc as 30 seconds) it will delete itself.
Thread* **thread**()	Returns the thread that handles data input and output.
const std::string& **type**() const	Retrieves the type of port specified by *type* in the constructor.
std::string& **user_agent**()	Returns the user agent assigned by calling set_proxy. There is no default value.
const std::string& **username_fragment**() const	A string that must match the port password to help ensure a globally unique identifier for this port. See **password**. A random default value is assigned on initialization.

Signals

SignalAddressReady sigslot::signal1< Port * >

Sent in reply to **PrepareAddress** when the port has generated a local address.

SignalConnectionCreated sigslot::signal2< Port *, Connection * >

Sent when a **Connection** object is added to the port. This is no guarantee that the connection is writable; you must connect to **Connection::SignalStateChange** to send STATE_WRITABLE before attempting to write to this connection.

SignalUnknownAddress sigslot::signal4< Port *, const SocketAddress &, StunMessage *, const std::string & >

Sent when the port received a successful STUN binding request from an address that isn't listed in its **Connection** list. To create this connection, the listener should call **CreateConnection** and then **SendBindingResponse**; otherwise call **SendBindingErrorResponse** to send an error message to the requester.

SignalReadError sigslot::signal2< Port *, int >

A read error occurred. This signal is not currently sent.

SignalWriteError sigslot::signal2< Port *, int >

A write error occurred. This signal is not currently sent.

SignalDestroyed sigslot::signal1< Port * >

Sent before the is destroyed (for example, it times itself out for lack of connections).

Friends

- Connection

Attributes: public

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