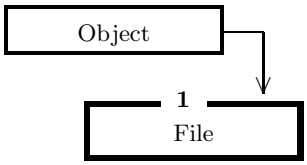
Contents

1

class **File** : public Object

File class.

Inheritance



Public Members

Protected Members

File class. This class encapsulates a C file. I wanted a way to do non-blocking I/O but could not find one using C++. So this class has been designed to do that. It can switch between blocking and non-blocking using the set_blocking() method. Everything else is reasonably straightforward.

Author: David Lapsley (dlapsley@haystack.mit.edu)

1.4

static const int **in**

Constant.

Constant. This constant is used to indicate that a new File object is is for reading (in).

1.5

static const int **out**

Constant.

Constant. This constant is used to indicate that a new File object is is for writing (out).

1.6

static const int **blocking**

Constant.

Constant. This constant is used to indicate the blocking mode of operation.

1.7

static const int **nonblocking**

Constant.

Constant. This constant is used to indicate the nonblocking mode of operation.

1.8

File (const string& name, const int& mode)

Constructor.

Constructor. The constructor opens the file with the supplied name and opens it in the appropriate mode (File::in, File::out).

Parameters:

`name`	The name of the file to be open.
`mode`	The mode to use in opening the file.

1.9

virtual **~File** ()

Destructor.

Destructor.

1.10

int **read** (char* b, const int& n)

I/O method Reads data from the file into the supplied buffer.

I/O method Reads data from the file into the supplied buffer.

Return Value: `Number` of bytes read.
Parameters: `b` Buffer to store data in.
`n` Number of bytes to attempt to read.

1.11

int **write** (const char* const b, const int& n)

I/O method Writes data from the supplied buffer to the file.

I/O method Writes data from the supplied buffer to the file.

Return Value: `Number` of bytes written.
Parameters: `b` Buffer to read data from.
`n` Number of bytes to attempt to write.

1.12

int **set_blocking** (const int& b)

Blocking method.

Blocking method. Sets the blocking mode of the file.

Parameters: `b` Blocking mode (File::blocking, File::nonblocking).

1.13

int **get_blocking** () const

Blocking method.

Blocking method. Get blocking mode.

Return Value: `the` blocking mode of the file.

1.14

int **close** ()

File connection termination.

File connection termination. Closes the file.

Return Value: `0:` on success, -1 on failure.

1.1

int **_fd**

File descriptor.

File descriptor. This member encapsulates the file descriptor.

1.2

int **_blocking**

Blocking flag.

Blocking flag. This indicated whether or not the File object is in blocking mode or not.

1.3

int **_blocking_flags**

Blocking flags.

Blocking flags. These are the flags returned from fcntl() prior to setting a File non-blocking. They are stored in this member so that they can be restored if a request to go back to blocking mode is made.

2

Socket module

Socket module This module implements base level socket functionality. It provides a number of key classes that are utilized by derived classes.

Author: David Lapsley (dlapsley@haystack.mit.edu)

3

struct **SocketBufferCtrlBlk**

Socket buffer control block.

Members

Socket buffer control block. Used to maintain state information along with the data itself.

3.1

int **_bytes_read**

Bytes read originally read into socket buffer(_sb).

Bytes read originally read into socket buffer(_sb).

3.2

int **_bytes_sent**

Bytes that have been transmitted from the buffer(_sb).

Bytes that have been transmitted from the buffer(_sb).

3.3

SocketBuffer **_sb**

Internal storage of data.

Internal storage of data.

3.4

SocketBufferCtrlBlk (const int size)

Constructor.

Constructor.

Parameters: `size` maximum number of bytes to store in buffer.

3.5 SocketBufferCtrlBlk ()

Default Constructor.

Default Constructor.

3.6 SocketBufferCtrlBlk (const SocketBufferCtrlBlk& s)

Copy Constructor.

Copy Constructor.

Parameters: `s` buffer to be iniitalized from.

3.7 SocketBufferCtrlBlk& operator= (const SocketBufferCtrlBlk& s)

Assignment operator

Assignment operator

Parameters: `s` buffer to be initialized from.

3.8

~SocketBufferCtrlBlk ()

Destructor

Destructor

3.9

int get_bytes_read () const

Return Value: `number` of bytes read.

3.10

int get_bytes_sent () const

Return Value: `number` of bytes sent.

3.11

const SocketBuffer& get_sb () const

Return Value: `reference` to socket buffer.

4

class **SocketException**

Used for trapping errors

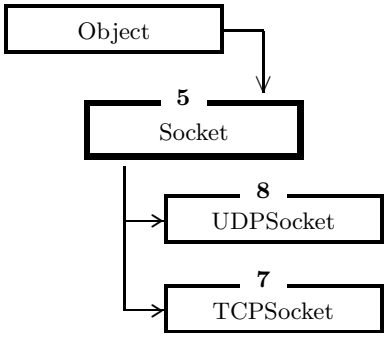
Used for trapping errors

5

class **Socket** : public Object

Base level socket class.

Inheritance



Public Members

Protected Members

Base level socket class. All other socket classes are derived from this one. This class inherits from the Object class.

5.18

static const int **blocking**

Constant flag value used in set_blocking().

5.19

static const int **nonblocking**

Constant flag value used in set_blocking().

5.20

Socket (const int& type)

Constructor.

Constructor. This constructor will create a socket of the supplied type by calling the C socket() function. constructs a Socket object.

Return Value: `none`

Parameters:

`type`	type of socket: currently either SOCK_STREAM or SOCK_STREAM.

5.21

Socket (const int& type, const int& s, const int&m)

Constructor.

Constructor. This constructor does NOT create a socket. It assigns the internal socket descriptor member (_sockd) to the supplied socket descriptor. constructs a Socket object.

Return Value: `none`

Parameters:

`type`	type of socket (as per C socket() call).
`s`	pre-opened socket descriptor (e.g. as returned from C socket() or accept() call).
`m`	socket Maximum Transmission Unit.

5.22

Socket ()

Default constructor.

Default constructor. Maximum Transmission Unit is set to 1024 by default. Socket type is set to SOCK_STREAM.

5.23

virtual ~**Socket** ()

Destructor.

Destructor. Important! The destructor does not close() the socket. This is because there are situations when it is necessary to have an opened socket survive the Socket object that created it. Users of the Socket class are responsible for closing the any sockets created using the close() or shutdown() methods.

5.24

int **get_sockd** () const

Access method.

Access method.

Return Value: `value` of sockd.

5.25

int **get_type** () const

Access method.

Access method.

Return Value: `value` of type.

5.26

int **get_mtu** () const

Access method.

Access method.

Return Value: `value` of mtu.

5.27

int **get_bytes_sent** () const

Access method.

Access method.

Return Value: `value` of bytes sent.

5.28

int **get_bytes_rcvd** () const

Access method.

Access method.

Return Value: `value` of bytes received.

5.29

int **get_send_calls** () const

Access method.

Access method.

Return Value: `value` of send calls.

5.30

int **get_recv_calls** () const

Access method.

Access method.

Return Value: `value` of recv calls.

5.31

int **set_so_reuseaddr** (const int& yes)

Access method.

Access method. set SO_REUSEADDR socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `yes` 0: allow address reuse, 1: disallow address reuse.

5.32

int **get_so_reuseaddr** () const

Access method.

Access method.

Return Value: `value` of so_reuseaddr.

5.33

int **set_so_linger** (const int& onoff, const int& time)

Access method.

Access method. set SO_LINGER socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `onoff` 0: disable linger option, 1: enable linger option.
`time` if linger enabled, how long to linger for.

5.34

conststruct linger& **get_so_linger** () const

Access method.

Access method.

Return Value: `value` of so_linger.

5.35

int **set_so_rcvbuf** (const int& sz)

Access method.

Access method. set SO_RCVBUF socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `sz` size to set so_rcvbuf.

5.36

int **get_so_rcvbuf** () const

Access method.

Access method.

Return Value: `value` of so_rcvbuf.

5.37

int **set_so_sndbuf** (const int& sz)

set SO_SNDBUF socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `sz` size to set so_sndbuf

5.38

int **get_so_sndbuf** () const

Access method.

Access method.

Return Value: `value` of so_sndbuf.

5.39

int **set_so_rcvlowat** (const int& m)

Access method.

Access method. set SO_RCVLOWAT socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `m` value to set rcv_lowat

5.40

int **get_so_rcvlowat** () const

Access method.

Access method.

Return Value: `value` of rcv_lowat.

5.41

int **set_so_sndlowat** (const int& m)

set SO_SNDLOWAT socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `m` size to set so_sndlowat

5.42

int **get_so_sndlowat** () const

Access method.

Access method.

Return Value: `value` of so_sndlowat.

5.43

int **set_tcp_nodelay** (const int& n)

Access method.

Access method. set TCP_NODELAY socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `n` value to set nodelay to. 0: allow delayed sending, 1: disallowdelayed sending.

5.44

int **get_tcp_nodelay** () const

Accessm method.

Accessm method.

Return Value: `value` of tcp_nodelay.

5.45

int **set_blocking** (const int& b)

Access method.

Access method. set O_NONBLOCK socket option

Return Value: `0:` on success, -1: on failure.
Parameters: `b` blocking mode: Socket::blocking: node is blocking (default),Socket::nonblocking: node is non-blocking.

5.46

int **get_blocking** () const

Access method.

Access method. get current blocking state o fthe socket

Return Value: `current` socket blocking mode.

5.47

const int **shutdown** (const int& how)

Access method.

Access method. shutdown connection as per standard socket library.

Parameters: `how` method to use to shutdown socket: SHUT_RD, SHUT_WR, SHUT_RDWR.

5.48

const int **close** ()

Access method.

Access method. close socket connection.

Return Value: `0:` on success, -1: on failure.

5.1

int **_sockd**

Encapsulated socket descriptor.

5.2

int **_type**

Socket type: SOCK_DGRAM(UDP) or SOCK_STREAM(TCP). Support for othe types may be added in the future.

5.3

int **_mtu**

Maximum Transmission Unit for this socket.

5.4

int **_bytes_sent**

Total number of bytes sent during the lifetime of this socket.

5.5

int **_bytes_rcvd**

Total number of bytes received during the lifetime of this socket.

5.6

int **_send_calls**

Total number of send calls made during the lifetime of this socket.

5.7

int **_recv_calls**

Total number of recv calls made during the lifetime of this socket.

5.8

int **_so_reuseaddr**

Current value of socket option.

5.9

struct linger **_so_linger**

Current value of socket option.

5.10

int **_so_rcvbuf**

Current value of socket option.

5.11

int **_so_sndbuf**

Current value of socket option.

5.12

int **_so_rcvlowat**

Current value of socket option.

5.13

int **_so_sndlowat**

Current value of socket option.

5.14

int **_tcp_nodelay**

Current value of tcp option.

5.15

int **_ip_tos**

Current value of ip option.

5.16

int **_blocking**

Current value of socket blocking option.

5.17 int _blocking_flags

Stored value of blocking flags.

6

TCPSocket class

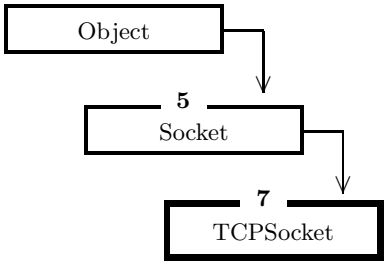
TCPSocket class This class encapsulates a TCP stream.

7

class **TCPSocket** : public Socket

TCPSocket class

Inheritance



Public Members

TCPSocket class This class encapsulates a TCP Stream. It inherits from the Socket class and provides methods that closely match those of the BSD sockets API. The main aim of this class is to simplify the creation, connection establishment and transmission/reception procedures of TCP sockets. This class has been deliberately designed to NOT handle errors for the calling context. Instead, it returns error codes back to the calling context so that it may work out how best to deal with it. The reason for doing this is that the TCPSocket does not have the wider context of the caller and so cannot determine if an error return is indeed an error or expected behavior.

7.1

TCPSocket ()

Default constructor.

Default constructor. This calls the base constructor to create an opened TCP socket (i.e. socket of type SOCK_STREAM)

7.2

TCPSocket (const int &m)

Constructor.

Constructor. This constructor creates a socket and sets the internal MTU member to be equal to the supplied parameter.

Parameters: `m` MTU for this socket.

7.3

TCPSocket (const TCPSocket &s)

Copy constructor.

Copy constructor. Copies values of other TCPSocket into this object.

Parameters: `s` TCPSocket to copy.

7.4

TCPSocket (const int& s, const int& m)

Constructor.

Constructor. This constructor is provided two initizialization parameters.

Parameters: `s` socket descriptor to encapsulate.
`m` MTU for this socket.

7.5

virtual **~TCPSocket** ()

Destructor.

Destructor.

7.6

TCPSocket& **operator=** (const TCPSocket& s)

Assigment operator.

Assigment operator. Copies supplied socket fields into this object.

Parameters: `s` socket to copy.

7.7

void **set_sockd** (const int& s)

Access method.

Access method. Allows setting of TCPSocket's internal socket descriptor. This is useful for allowing a TCPSocket object to encapsulate a socket descriptor that has been returned from another function, e.g. accept().

Parameters: `s` socket descriptor to encapsulate.

7.8

int **bind** (const string& ip, const int& port)

Socket API method.

Socket API method. Binds socket to supplied address.

Return Value: `0:` on success, -1 on failure.
Parameters: `ip` IP address to bind to.
`port` port to bind to.

7.9

int **listen** ()

Socket API method.

Socket API method. Prepare a socket for accepting incoming connections.

Return Value: `0:` on success, -1 on failure.

7.10

int **accept** ()

Socket API method.

Socket API method. Wait for incoming connections on a socket.

Return Value: `-1:` on failure, positive integer on success. On succesthe return value is a valid socket descriptor.

7.11

int **connect** (const string& ip, const int& port)

Socket API method.

Socket API method. Attempt to establish a connection to the supplied address.

Return Value: `0:` on success, -1 on failure.
Parameters: `ip` IP address to connect to.
`port` port to connect to.

7.12

int **recv** (SocketBuffer& s, const int& n)

Socket API method.

Socket API method. Attempt to receive data into supplied buffer.

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes read.
Parameters: `s` socket buffer to store data in.
`n` number of bytes to attempt to receive.

7.13

int **recv** (SocketBuffer& s, const int& pos, const int& n)

Socket API method.

Socket API method. Attempt to recieve data into supplied buffer.

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes read.

Parameters:
- `s` socket buffer to store data in.
- `pos` starting position in the supplied buffer (s) at which tostore received data.
- `n` number of bytes to atttempt to receive.

7.14

int **send** (const SocketBuffer& s, const int& n)

Socket API method.

Socket API method. Attempt to send data from supplied buffer.

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes sent.

Parameters:
- `s` socket buffer to from which to get data to send.
- `n` number of bytes to attempt to send.

7.15

int **send** (const SocketBuffer& s, const int& pos, const int& n)

Socket API method.

Socket API method. Attempt to send data from supplied buffer.

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes sent.

Parameters:

`s`	socket buffer to from which to get data to send.
`pos`	starting position in the supplied buffer (s) at which tostart transmitting data.
`n`	number of bytes to attempt to send.

7.16

int **shutdown** (int howto)

Socket API method.

Socket API method. shutdown connection as per standard socket library.

Parameters:

`how`	method to use to shutdown socket: SHUT_RD, SHUT_WR, SHUT_RDWR.

7.17

int **close** ()

Socket API method.

Socket API method. close socket connection.

Return Value:

`0:`	on success, -1: on failure.

7.18

static bool **test** ()

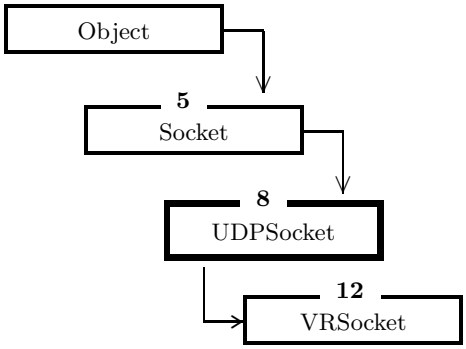
Test method.

8

class **UDPSocket** : public Socket

UDPSocket class.

Inheritance



Public Members

UDPSocket class. This class encapsulates a UDP packet stream. It inherits from the socket class and provides methods that closely match those of the BSD sockets API. The main aim of this class is to simplify the creation and data transmission/reception procedures of UDP sockets. This class has been * deliberately designed to NOT handle errors for the calling context. Instead, it returns error codes back to the calling context so that it may work out how best to deal with it. The reason for doing this is that the TCPSocket does not have the wider context of the caller and so cannot determine if an error return is indeed an error or expected behavior.

8.1

UDPSocket ()

Default constructor.

Default constructor. This calls the base constructor to create an opened UDP socket (i.e. socket of type SOCK_DGRAM)

8.2

UDPSocket (const int& s, const int& m)

Constructor.

Constructor. This constructor is provided two initizialization parameters.

Parameters: `s` socket descriptor to encapsulate.
`m` MTU for this socket.

8.3

UDPSocket (const UDPSocket& s)

Copy constructor.

Copy constructor. Copies values of other UDPSocket into this object.

Parameters: `s` UDPSocket to copy.

8.4

virtual ~**UDPSocket** ()

Destructor.

Destructor.

8.5

UDPSocket& **operator=** (const UDPSocket& s)

Assigment operator.

Assigment operator. Copies supplied socket fields into this object.

Parameters: `s` socket to copy.

8.6

void **set_sockd** (const int& s)

Access method.

Access method. Allows setting of TCPSocket's internal socket descriptor. This is useful for allowing a TCPSocket object to encapsulate a socket descriptor that has been returned from another function, e.g. accept().

Parameters: `s` socket descriptor to encapsulate.

8.7

void **bind** (const string& ip, const int& port)

Socket API method.

Socket API method. Binds socket to supplied address.

Return Value: `0:` on success, -1 on failure.
Parameters: `ip` IP address to bind to.
`port` port to bind to.

8.8

void **connect** (const string& ip, const int& port)

Socket API method.

Socket API method. Attempt to establish a connection to the supplied address. Note that UDP is connectionless and that in this case "connection" refers to the fact that the destination address fields are set to the parameters supplied in order that subsequent calls to send() use these parameters. Otherwise, an address must be supplied with each call to sendto().

Return Value: `0:` on success, -1 on failure.
Parameters: `ip` IP address to connect to.
`port` port to connect to.

8.9

int **recv** (SocketBuffer& s, const int& n)

Socket API method.

Socket API method. Attempt to receive data into supplied buffer.

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes read.
Parameters: `s` socket buffer to store data in.
`n` number of bytes to attempt to receive.

8.10

int **send** (const SocketBuffer& s, const int& n)

Socket API method.

Socket API method. Attempt to send data from supplied buffer. In this case, the destination is the address specified in the last call to connect().

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes sent.
Parameters: `s` socket buffer to from which to get data to send.
`n` number of bytes to attempt to send.

8.11

int **sendto** (const string& ip, const int& port, const Socket-Buffer& s, const int& n)

Socket API method.

Socket API method. Attempt to send data from supplied buffer to specified destination.

Return Value:
- `0:` on success, positive integer on success. On successthe value returned is the number of bytes sent.

Parameters:
- `ip` IP address to send data to.
- `port` port to send data to.
- `s` socket buffer to from which to get data to send.
- `n` number of bytes to attempt to send.

8.12

void **close** ()

Socket API method.

Socket API method. close socket connection.

Return Value:
- `0:` on success, -1: on failure.

8.13

static bool **test** ()

Test method.

9

typedef deque<SocketBufferCtrlBlk> **SocketBufferCtrlBlkList**

A deque of socket buffer control blocks

A deque of socket buffer control blocks

10

typedef map<int, SocketBufferCtrlBlk> **SocketBufferCtrlBlkMap**

This is used to hold SocketBufferCtrlBlks that have been transmitted but are still pending acknowledgment

This is used to hold SocketBufferCtrlBlks that have been transmitted but are still pending acknowledgment

11

VRSocket class

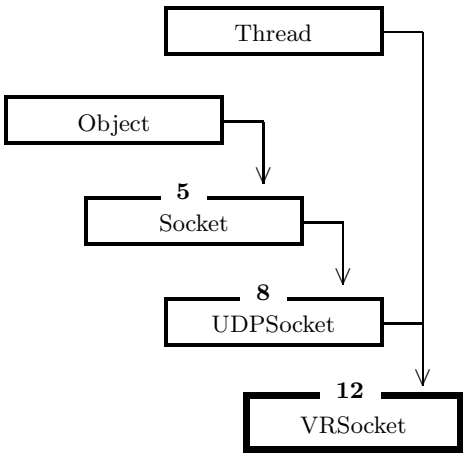
VRSocket class This class encapsulates a UDP stream and implements the VLBI Real-time protocol on top of it.

12

class **VRSocket** : public UDPSocket, public Thread

VRSocket class This class encapsulates a socket processing thread.

Inheritance



Public Members

Private Members

VRSocket class This class encapsulates a socket processing thread. It spawns a new thread of execution for asynchronously processing send and receive requests. This class is responsible for reading from/writing to a socket. Once "thread_create()" is called, the VRSocket object will spawn a thread, which in turn will execute the thread_func() method. This method continually reads/writes to a socket. Data is read from/written to the send/recv buffers.

12.11

static const int **receiver**

Mode constant.

Mode constant.

12.12

static const int **sender**

Mode constant.

Mode constant.

12.13

VRSocket (const int& mode)

Default constructor.

Default constructor. This calls the base constructor to create a UDP socket and initialize a thread to service it.

Parameters:

`mode`	The mode for this socket (either send or receive).

12.14

VRSocket (const int& mode, const int &m)

Constructor.

Constructor. This constructor creates a socket and sets the internal MTU member to be equal to the supplied parameter.

Parameters:

`mode`	The mode for this socket (either send or receive).
`m`	MTU for this socket.

12.15

VRSocket (const VRSocket &s)

Copy constructor.

Copy constructor. Copies values of other VRSocket into this object.

Parameters:

`mode`	The mode for this socket (either send or receive).
`s`	VRSocket to copy.

12.16

VRSocket (const int& mode, const int& s, const int& m)

Constructor.

Constructor. This constructor is provided three initialization parameters.

Parameters:

`mode`	The mode for this socket (either send or receive).
`s`	socket descriptor to encapsulate.
`m`	MTU for this socket.

12.17

virtual **~VRSocket** ()

Destructor.

Destructor.

12.18

VRSocket& **operator=** (const VRSocket& s)

Assigment operator.

Assigment operator. Copies supplied socket fields into this object.

Parameters: `s` socket to copy.

12.19

int **recv** (SocketBuffer& s, const int& n)

Access method.

Access method. Allows setting of VRSocket's internal socket descriptor. This is useful for allowing a VRSocket object to encapsulate a socket descriptor that has been returned from another function, e.g. accept().

Parameters: `s` socket descriptor to encapsulate.

12.20

int **send** (const SocketBuffer& s, const int& n)

Socket API method.

Socket API method. Attempt to recieve data into supplied buffer.

Return Value: `0:` on success, positive integer on success. On successthe value returned is the number of bytes read.

Parameters:
- `s` socket buffer to store data in.
- `pos` starting position in the supplied buffer (s) at which tostore received data.
- `n` number of bytes to atttempt to receive.

12.21

int **get_send_buf_size** () const

Access method.

Access method. Returns the size of send_buf (in number of buffers). This provides a method for the caller to determine the progress of their request.

12.22

static bool **test** ()

Test method.

12.1

Data storage.

Data storage. Stores data that is to be transmitted (in "send" mode) or received (in "recv" mode). Note that a mutex is not required since the object is only in one mode or another. Although the object will be sending and receiving information, data flow is restricted to one direction, and control the other.

12.2

int **_mtu**

MTU size.

MTU size. MTU size for data transmission, receptions.

12.3

void* **thread_func** ()

Thread callback.

Thread callback. This function is called when the base class method "thread_create()" is called. This results in thread_func() being executed in it's own thread. Depending on the mode, either recv_proc() or send_proc() will be called by thread_func(). This means that recv_proc() or send_proc() will be running in their own thread of execution.

12.4

void **recv_proc** ()

Processing function.

Processing function. This function does all of the processing when the object is in recv mode.

12.5

void **send_proc** ()

Processing function.

Processing function. This function does all of the processing when the object is in send mode.

12.6

SocketBufferCtrlBlkList **_recv_sockbufctrlblk_list**

Used for storing buffers containing received data.

12.7

SocketBufferCtrlBlkList **_send_sockbufctrlblk_list**

Used for storing buffers for transmission.

12.8

SocketBufferCtrlBlkMap **_outstanding_sockbufctrlblks**

Used for storing buffers that have been transmitted, but not yet acknowledged.

12.9

Mutex **_send_mutex**

Mutex.

Mutex. Controls multi-threaded access to send data members.

12.10

Mutex **_recv_mutex**

Mutex.

Mutex. Controls multi-threaded access to rcv data members.

Class Graph