Contents

1

#define **RMBX_MAILBOX_ID_SIZE**

Mailbox identifier maximum length.

Mailbox identifier maximum length. This is the maximum allowed length for mailbox identifier strings. This constant is simply used to dimension internal buffers which receive the mailbox identifiers, and therefore can be simply set to a higher value without other modifications, no other limits are imposed.

2

#define **RMBX_MAX_MESSAGE_SIZE**

Message body maximum length.

Message body maximum length. This is the maximum allowed length for the message body (therefore not including source mailbox identifier, string terminator, message length bytes). This constant is currently used to dimension internal buffers which receive the mailbox identifiers, but it cannot be set to a higher value without other modifications, since currently two bytes are used to indicate the message body length.

3

#define **RMBX_SERVICE_PORT**

Mailbox server port.

Mailbox server port. This is the standard port on which the mailboxd server is bound.

4

#define **RMBX_DEFAULT_VERBOSITY**

Default verbosity level.

Default verbosity level. This is the default verbosity level for the object belonging to the CRemoteMailbox class. It is always possible to override this setting by using the <I>SetVerbosity()</I> mathod.

5 class **CRemoteMailbox**

Remote mailbox client class.

Public Members

Protected Members

Remote mailbox client class. This class represents a mailbox that is stored and managed by a remote host on the behalf of the process that owns the mailbox itself and the object of this class which is associated to it.

This class exports a number of methods that can be used to connect as a client to a mailboxd daemon on a networked server, to register onto it with an unique identifier and create a mailbox on it, to write a message to another mailbox and to read the mailbox.

5.6

bool **BroadcastMessage** (char * strText, int iLen)

Sends a to all the registered mailbox.

Sends a to all the registered mailbox. This function sends a message to all the mailbox that have been registered onto the mailboxd daemon, waiting for an acknowledgment return message from the server; due to this feature, the call could be blocking. Note that no destination mailbox parameter is requested.

Return Value: `true` if the message has been successfully delivered; false otherwise (this is a common error when this mailbox client or the destination counterpart is not connected to the server, or the destination mailbox name was misspelled;

Parameters:
- `strText` this is the buffer containing the message that has to be delivered; this string is limited in length to the value of the RMBX_MAX_MESSAGE_SIZE constant. Since the lenght of this buffer is separately indicated, it does not need to be a zero-terminated string, but can contain any character.
- `iLen` contains the lenght of the message text buffer.

See Also: SendMessage();
RMBX_MAX_MESSAGE_SIZE (→2, *page 3*)

5.7

CRemoteMailbox ()

Standard class constructor.

Standard class constructor. Simply initializes member variables to null or default values. It sets the verbosity to the current value of RMBX_DEFAULT_VERBOSITY, as it can be found in remotemailbox.h

See Also: RMBX_DEFAULT_VERBOSITY
~CRemoteMailbox();

5.8

bool **Connect** (char * strMailboxName, char * strServerHostName, int iPort)

Connects to server.

Connects to server. This function connects to the specificed host, trying to register a mailbox with the desired name.

Return Value: `true` if the connection has been successfully set up; false if it was not able to connect or there is another connected participant with the desired name.

Parameters:

`strMailboxName`	this is the name of the mailbox for your process you want to create.
`strServerHostName`	this is the name of the server you want to connect to; this argument has the default value "localhost".
`iPort`	this is the server port number to which you want to connect; this argument has a default value of RMBX_SERVICE_PORT, which is originally set to 12345.

See Also: IsConnected();

5.9

void **Disconnect** ()

Disconnects from the server.

Disconnects from the server. Closes the communication socket, causing the server to remove the registered mailbox and to delete possible non-read messages. The operation has no effects if the socket was not connected.

See Also: m_iSocket * @see IsConnected();

5.10

bool **IsConnected** ()

Returns the connection status.

Returns the connection status. This method returns the current status of the communication.

Return Value: s true if the mailbox-associated socket is connected to the mailboxd daemon.

See Also: Connect();

5.11

bool **ReceiveMessage** (char * strSender, int* pnchSender, char * strMsgBody, int* pnchMsgBody)

Receives a message.

Receives a message. This function receives the first message that has been enqueued in the mailbox registered onto the mailboxd daemon by the calling process. If no messages are waiting, the call is blocking.

Return Value:	`true`	if the message has been successfully retrieved; false otherwise (this is a common error when the supplied buffers are not large enough to receive the data
Parameters:	`iLen`	contains the lenght of the message text buffer.
See Also:	SendMessage();	

5.12

bool SendMessage (char * strDestination, char * strText, int iLen)

Sends a message to a mailbox.

Sends a message to a mailbox. This function sends a message to the desired destination mailbox, waiting for an acknowledgment return message from the server; due to this feature, the call could be blocking.

Return Value:	`true`	if the message has been successfully delivered; false otherwise (this is a common error when this mailbox client or the destination counterpart are not connected to the server, or the destination mailbox name was misspelled);
Parameters:	`strDestination`	this is the name of the destination mailbox; this string is limited to RMBX_MAILBOX_ID_SIZE characters in length.
	`strText`	this is the buffer containing the message that has to be delivered; this string is limited in length to the value of the RMBX_MAX_MESSAGE_SIZE constant. Since the lenght of this buffer is separately indicated, it does not need to be a zero-terminated string, but can contain any character.
	`iLen`	contains the lenght of the message text buffer.

See Also: BroadcastMessage();
RMBX_MAX_MESSAGE_SIZE
RMBX_MAILBOX_ID_SIZE (→1, *page 2*)

5.13

void **SetVerbosity** (bool bVerbose)

Sets the verbosity level.

Sets the verbosity level. Use this function to override the default value of the verbosity. When the verbosity level is true, the object sends results and remarks to the process-associated terminal about the object's ongoing activities;

Parameters: `bVerbose` set this value to true if you want to see results and remarks on the object's ongoing activities; set it to false to disable them.

See Also: m_bVerbose

5.14

~CRemoteMailbox ()

Standard class destructor.

Standard class destructor. Simply closes the associated socket, to disconnect pending communications, if any.

See Also: Disconnect();

5.1

int **m_iSocket**

Socket file descriptor number.

Socket file descriptor number. This is the number of the file descriptor associated to the socket that is currently in use. This number is meaningless if the socket is not actually connected; this member variable is stored for internal use only, its value is determined by the system at run-time.

See Also: m_bConnected
IsConnected (→5.10, *page 9*)

5.2

bool **m_bConnected**

Connection flag.

Connection flag. This flag indicates whether the socket is connected or not. This member variable is stored for internal use only, callers should read its value by using the <I>IsConnected()</I> method.

See Also: IsConnected (→5.10, *page 9*)

5.3

bool **m_bVerbose**

Verbosity level.

Verbosity level. This flag indicates whether the program should generate information and progress status messages during the program execution. Messages are sent to the associated terminal. This flag is initially set to the default verbosity level; this member variable is stored for internal use only, use instead the <I>SetVerbosity()</I> method to modify its value.

See Also: SetVerbosity (→5.13, *page 11*)

5.4

char **m_strMailBoxName** [RMBX_MAILBOX_ID_SIZE+1]

Mailbox identifier.

Mailbox identifier. This is the currently chosen mailbox identifier. This data member is meaningless is the mailbox is not connected. This member variable is stored for internal use only, use instead the <I>Connect()</I> method to set its value at connection time.

See Also: IsConnected (→5.10, *page 9*)

5.5

struct sockaddr_in **m_sinAddress**

Server address.

Server address. This structure contains the inet address with host and port of the server on which the mailboxd server is running. This member variable is stored for internal use only.

6 Mailbox State identifiers (x)

State identifiers.

Names

#define **STATE_UNNAMED**
State identifier: the mailbox has not yet been initialised

#define **STATE_EXTR_NAME**
State identifier: the mailbox is reading its name

#define **STATE_WAITING**
State identifier: the mailbox is idle

#define **STATE_EXTR_DEST**
State identifier: the mailbox is receiving the destination for an incoming message

#define **STATE_EXTR_LEN1**
State identifier: the mailbox is receiving the low-order byte of the length word for an incoming message

#define **STATE_EXTR_LEN2**
State identifier: the mailbox is receiving the high-order byte of the length word for an incoming message

#define **STATE_EXTR_BODY**
State identifier: the mailbox is receiving the body of an incoming message

#define **STATE_ACKNOWLEDGING**
State identifier: the mailbox is acknowledging the sender about the completion of the message delivery

#define **STATE_SENDING**

State identifier: the mailbox is sending a message to the attached socket

State identifiers. These are the identifiers used by the finite state machine implemented by the mailbox.

7 class **CMailbox**

Mailbox server class implementation.

Public Members

		CMailbox ()	*Default constructor. Initializes data to empty values.*	
		Setup (int iSocket)	*Initializes a new mailbox. Associates an unused mailbox to a socket.*	
		~CMailbox ()	*Default destructor.*	
7.1		**Close** ()	*Close the mailbox.*	18
7.2	bool	**IsUsed** ()	*Returns true if the mailbox is in use.*	18
7.3	const char*	**GetName** ()	*Returns the mailbox name.*	18
7.4	const int	**GetSocket** ()	*Returns the associated socket.* ..	18
7.5	const char*	**GetMsgDest** ()	*Returns the message recipient.* .	19
7.6	const int	**GetMsgLen** ()	*Returns the message body length.* 19	
	const char*	**GetMsgBody** ()	*Returns the incoming message body text*	
7.7	void	**Delivered** (bool bSuccess)	*Indicates the delivery success.* ..	19
7.8	bool	**HasPendingOutput** ()	*Returns true if the socket has pending output.*	20
7.9	void	**EnqueueData** (const char * strData, int iLen)	*Enqueue output data.*	20
7.10	bool	**WriteEnqueuedData** ()	*Write outgoing data.*	20
7.11	bool	**CharReceived** (unsigned char ch)	*Notifies mailbox of a character arrival.*	20

Protected Members

bool	**m_bIsUsed**	*Use flag. Indicates whether the mailbox is in use or not.*
char	**m_strName** [ID_BUF_SIZE+1]	*Malbox Name. Contains the name of the mailbox.*
int	**m_iSocket**	*Socket file descriptor number. This is the file descriptor number of the socket associated to the mailbox.*
int	**m_iState**	*Mailbox state. This is the current state of the finite state machine which implements the mailbox.*
int	**m_iPosition**	*Current input buffer position. This is the current position in the input buffer of the mailbox, it is u*
int	**m_iMsgLen**	*Input message length. This is the read lenght of the body of the message which is being read.*
int	**m_iOutBufferSize**	*Output buffer size. This is the size of the allocated output data buffer*
int	**m_iOutBufferLen**	*Used portion of the output data buffer*
char	**m_strDest** [ID_BUF_SIZE+1]	*Recipient mailbox identifier.*
char	**m_strBody** [TXT_BUF_SIZE+1]	*Text of the incoming message*
char*	**m_strOutBuffer**	*Dynamically allocated pending output data buffer*

Mailbox server class implementation.

7.1

Close ()

Close the mailbox.

Close the mailbox. Resets the mailbox associated data and indicates that the mailbox is unused and can be set up for new mailboxes.

7.2

bool **IsUsed** ()

Returns true if the mailbox is in use.

Returns true if the mailbox is in use. The mailbox is defined to be in use when it has an associated socket and a remote process connected to it, even if the remote process has not yet indicatet its name.

7.3

const char* **GetName** ()

Returns the mailbox name.

Returns the mailbox name. Returns the name of the mailbox which has been chosen by the remote process connected to it.

7.4

const int **GetSocket** ()

Returns the associated socket.

Returns the associated socket. Returns the file descriptor of the socket currently associated to the mailbox.

7.5

const char* **GetMsgDest** ()

Returns the message recipient.

Returns the message recipient. Returns the destination mailbox of the incoming message, if any. When the mailbox is not receiving any message (i.e. it is in the STATE
WAITING state), this is the destination of the last incoming message.

7.6

const int **GetMsgLen** ()

Returns the message body length.

Returns the message body length. Returns the length of the body (therefore not counting recipient mailbox identifier, string terminator, length bytes) of the incoming message, if any. When the mailbox is not receiving any message (i.e. it is in the STATE_WAITING state), this is the body length of the last incoming message.

7.7

void **Delivered** (bool bSuccess)

Indicates the delivery success.

Indicates the delivery success. Indicates to the mailbox that the last incoming message has been (or not) correctly delivered to destination mailbox. This method notifies the success (or insuccess) to the process owning the mailbox.

7.8

bool **HasPendingOutput** ()

Returns true if the socket has pending output.

Returns true if the socket has pending output. Returns true when the output pending data buffer is not empty. This represent a sort of request for the caller to call the WriteEnqueuedData() member as soon as the associated socket is ready for writing.

7.9

void **EnqueueData** (const char * strData, int iLen)

Enqueue output data.

Enqueue output data. Appends outgoing data to the the mailbox output buffer.

7.10

bool **WriteEnqueuedData** ()

Write outgoing data.

Write outgoing data. Tries to write to the socket the outgoing pending data, through a non-blocking write() call.

7.11

bool **CharReceived** (unsigned char ch)

Notifies mailbox of a character arrival.

Notifies mailbox of a character arrival. Updates the mailbox finite state machine with a new input

8 class **CMailboxCollection**

Mailbox collection class.

Public Members

		CMailboxCollection ()	*Default constructor. Initializes the mailbox collection with empty data.*	
		˜CMailboxCollection ()	*Default destructor. Deletes all the contained mailbox objects and frees the used memory.*	
8.4	int	**FindMailbox** (const char * strName, int iStartPos=0)	*Finds a mailbox by name.*	22
8.5	int	**FindMailbox** (int iSocket)	*Finds a mailbox by socket.*	22
8.6	void	**UpdateWriteSet** (fd_set * write_fd_set)	*Updates the socket write set.*	23
8.7	bool	**AddMailbox** (int iSocket)	*Adds a new mailbox.*	23
8.8	void	**WriteEnqueuedData** (int iSocket)	*Writes the enqueued data for the given socket.*	23
8.9	bool	**CloseMailbox** (const char * strName)	*Closes the mailbox which has a given name.*	24
8.10	bool	**CloseMailbox** (int iSocket)	*Closes the mailbox which has a given name.*	24
8.11	void	**CloseMailboxes** ()	*Closes all the mailboxes*	24
	bool	**CharFromSocket** (int iSocket, unsigned char ch)	*Notify character. Notify the mailbox associated to given socket of a character arrival.*	

Protected Members

Mailbox collection class.

8.4

int **FindMailbox** (const char * strName, int iStartPos=0)

Finds a mailbox by name.

Finds a mailbox by name. Returns the number of the first mailbox which has the given name, starting from the given point.

Return Value: `the` number of the mailbox which has the given name, or the value -1 if no mailboxes with the given name can be found.

Parameters:

`strName`	this is the name of the mailbox you want to find.
`iStartPos`	the number to start from in the search.

8.5

int **FindMailbox** (int iSocket)

Finds a mailbox by socket.

Finds a mailbox by socket. Returns the number of the mailbox which is associated to the given socket.

Return Value: `the` number of the mailbox which has the given name, or the value -1 if no mailboxes with the given name can be found.

Parameters: `iSocket` this is the socket associated to the mailbox you want to find.

8.6

void **UpdateWriteSet** (fd_set * write_fd_set)

Updates the socket write set.

Updates the socket write set. Updates the set of sockets which should be kept under observation to flush pending outgoing data. To achieve this, every used mailbox in the collection is examined to see whether it has pending data.

8.7

bool **AddMailbox** (int iSocket)

Adds a new mailbox.

Adds a new mailbox. Sets a new mailbox up with the given socket number. If every allocated mailbox is used, a new set of empty mailbox (of cardinality ALLOC
GRANULARITY) is allocated.

8.8

void **WriteEnqueuedData** (int iSocket)

Writes the enqueued data for the given socket.

Writes the enqueued data for the given socket. Identifies the socket in the collection which has the given socket and, using its WriteEnqueuedData method, asks it to write (if possible) pending outgoing data.

8.9

bool **CloseMailbox** (const char * strName)

Closes the mailbox which has a given name.

Closes the mailbox which has a given name. The mailbox with the given name is reinitialized, every pending data is lost.

8.10

bool **CloseMailbox** (int iSocket)

Closes the mailbox which has a given name.

Closes the mailbox which has a given name. The mailbox with the given socket is reinitialized, every pending data is lost.

8.11

void **CloseMailboxes** ()

Closes all the mailboxes

Closes all the mailboxes

8.1

CMailbox** **m_ppmbx**

Internal mailbox array.

Internal mailbox array. This data member points to a dynamically allocated area which contains the pointers to all the allocated mailbox objects.

8.2

int **m_iAllocated**

Number of allocated mailboxes.

Number of allocated mailboxes. This is the number of dynamically allocated mailbox objects.

8.3

int **m_iUsed**

Number of used mailboxes.

Number of used mailboxes. This is the count of the mailboxes currently in use, that is to say the mailboxes which have a name and an associated process and socket.

9 CMailboxCollection **mbxcoll**

Mailbox collection instance

Mailbox collection instance

10

void **ctl_c_catch** (int i)

SIGINT signal catcher.

SIGINT signal catcher. Intercepts SIGINT signals $(^C), ineventofitclosesallsocketsandterminatesprogram.$

11

int **main** ()

Mailboxd daemon main().

Mailboxd daemon main().

12

int **main** (int argc, char** argv)

Reader test program's main() function.

Reader test program's main() function. This function connects to a user-specified host, on which the mailboxd daemon should be up and running, and tries to register a test mailbox called "mailbox", reporting possible errors. The server hostname can be specified as the first parameter on the command line.

13

int **main** (int argc, char** argv)

Writer test program's main() function.

Writer test program's main() function. This function connects to a user-specified host, on which the mailboxd daemon should be up and running, and tries to send a message to a test mailbox called "mailbox", reporting possible errors. The server hostname can be specified as the first parameter on the command line.

Class Graph