Table of Contents

Chapter One: Introduction
INTRODUCTION TO SMB 5
INTRODUCTION TO THE RTSMB CLIENT 5
INTRODUCTION TO THE SMB PROTOCOL 5
BASICS 5
PROTOCOL HIERARCHY 5
BASIC NETWORKING 5
FLOW OF SMB COMMUNICATION 5
EXAMPLE 5
MODULES 6
PORTING 6
Instructions for porting using Visual C++: 6
Chapter Two: Configuring SMB
Parameters located in clicfg.c. 9
CFG_RTSMB_CLI_MAX_SESSIONS 9
CFG_RTSMB_CLI_MAX_SEARCHES_PER_SESSION 9
CFG_RTSMB_CLI_MAX_FILES_PER_SEARCH 9
CFG_RTSMB_CLI_MAX_FIDS_PER_SESSION 9
CFG_RTSMB_CLI_MAX_SHARES_PER_SESSION 9
CFG_RTSMB_CLI_MAX_SHARES_PER_SEARCH 9
CFG_RTSMB_CLI_MAX_SERVER_SEARCHES 9
CFG_RTSMB_CLI_MAX_SERVERS_PER_SEARCH 9
CFG_RTSMB_CLI_BUFFER_SIZE 9
CFG_RTSMB_CLI_MAX_BUFFERS_PER_WIRE 9
CFG_RTSMB_CLI_MAX_JOBS_PER_SESSION 9
CFG_RTSMB_CLI_MAX_SUPPORTED_THREADS 9
Chapter Three: Porting SMB
RTSMB Porting: Overview 13
Instructions for Porting Using Visual C++: 13
RTSMB Porting: Periodic Clock Support (psmbos.c) 13
Functions necessary to provide mutex semaphore support for RTSMB: 13
int rtsmb_osport_create_mutex (unsigned long *mutexHandle) 13
void rtsmb_osport_claim_mutex (unsigned long mutexHandle) 13
void rtsmb_osport_release_mutex (unsigned long mutexHandle) 13
RTSMB Porting: Thread Support (psmbos.c) 13
int rtsmb_osport_create_thread (RTSMB_THREAD_FN fn, void *context) 13
void rtsmb_osport_exit_thread (void) 13
RTSMB Porting: Network Support (psmbnet.c) 13
Functions that must be provided as an interface to the network stack 14
int rtsmb_netport_init (void) 14
int rtsmb_netport_accept 14
int rtsmb_netport_allow_broadcast 14
int rtsmb_netport_bind 14
int rtsmb_netport_closesocket 14
int rtsmb_netport_connect 15
int rtsmb_netport_listen 15
long rtsmb_netport_recv 15
long rtsmb_netport_recvfrom 15
int rtsmb_netport_select_n_for_read 15
long rtsmb_netport_send 16
long rtsmb_netport_sendto 16
int rtsmb_netport_socket_stream 16
int rtsmb_netport_socket_datagram 16
Chapter Four: SMB Client API
RTSMB_CLI_INIT 20
RTSMB_CLI_SHUTDOWN 20
RTSMB_CLI_SESSION_NEW_WITH_NAME 21
RTSMB_CLI_SESSION_CLOSE 21
RTSMB_CLI_SESSION_RESTART 22
RTSMB_CLI_SESSION_SET_BLOCKING 22
RTSMB_CLI_SESSION_CYCLE 23
RTSMB_CLI_SESSION_SET_JOB_CALLBACK 23
RTSMB_CLI_SESSION_LOGON_USER 24
RTSMB_CLI_SESSION_LOGOFF_USER 24
RTSMB_CLI_SESSION_CONNECT_SHARE 25
RTSMB_CLI_SESSION_DISCONNECT_SHARE 25
RTSMB_CLI_SESSION_SHARE_FIND_FIRST 26
RTSMB_CLI_SESSION_SHARE_FIND_NEXT 26
RTSMB_CLI_SESSION_SHARE_FIND_CLOSE 27
RTSMB_CLI_SESSION_OPEN 27
RTSMB_CLI_SESSION_CLOSE 28
RTSMB_CLI_SESSION_READ 28
RTSMB_CLI_SESSION_WRITE 29
RTSMB_CLI_SESSION_SEEK 29
RTSMB_CLI_SESSION_TRUNCATE 30
RTSMB_CLI_SESSION_FLUSH 30
RTSMB_CLI_SESSION_RENAME 31
RTSMB_CLI_SESSION_DELETE 31
RTSMB_CLI_SESSION_MKDIR 32
RTSMB_CLI_SESSION_RMDIR 32
RTSMB_CLI_SESSION_FIND_FIRST 33
RTSMB_CLI_SESSION_FIND_NEXT 33
RTSMB_CLI_SESSION_FIND_CLOSE 34
RTSMB_CLI_SESSION_STAT 34
RTSMB_CLI_SESSION_CHMODE 35
RTSMB_CLI_SESSION_GET_FREE 35
EZ API 36
RTSMB_CLI_EZ_SET_USER 36
RTSMB_CLI_EZ_OPEN 37
RTSMB_CLI_EZ_READ 37
RTSMB_CLI_EZ_WRITE 38
RTSMB_CLI_EZ_SEEK 38
RTSMB_CLI_EZ_CLOSE 39
RTSMB_CLI_EZ_TRUNCATE 39
RTSMB_CLI_EZ_FLUSH 40
RTSMB_CLI_EZ_RENAME 40
RTSMB_CLI_EZ_DELETE 41
RTSMB_CLI_EZ_MKDIR 41
RTSMB_CLI_EZ_RMDIR 42
RTSMB_CLI_EZ_FIND_FIRST 42
RTSMB_CLI_EZ_FIND_NEXT 43
RTSMB_CLI_EZ_FIND_CLOSE 43
RTSMB_CLI_EZ_FIND_STAT 44
RTSMB_CLI_EZ_FIND_CHMODE 44
RTSMB_CLI_EZ_GET_FREE 45
RTSMB_CLI_EZ_GET_CWD 45
RTSMB_CLI_EZ_SET_CWD 46
Appendix A: Examples:
EZ Layer Code 47
ENUMERATING SERVERS 49
ENUMERATING SHARES 49
ENUMERATING FILES 50
FILE I/O 50
Appendix B: Examples:
Synchronous Session Layer Code 53
ENUMERATING SERVERS 55
ENUMERATING SHARES 56
ENUMERATING FILES 56
FILE I/O 57
Appendix C: Examples:
Asynchronous Session Layer Code 59
ENUMERATING SERVERS 61
ENUMERATING SHARES 62
ENUMERATING FILES 63
FILE I/O 64
Appendix D: Error Values 67
The EZ API layer error values 69
Session API layer error values 69

RTIP 4.0

SMB Client Manual

Chapter One: Introduction

CHAPTER 1

Revised July 18, 2003

INTRODUCTION TO SMB

INTRODUCTION TO THE RTSMB CLIENT

A Windows network using SMB will be a group of computers, some running SMB servers, some running SMB clients, and some running both. A server simply sits and waits for clients to demand services. A client, however, must seek out servers on the network to which it can connect. Thus, there are two aspects to an SMB client. One is the meta-session details — finding out who exists on the network; the other is the session details — connecting, reading, writing, etc.

SMB name space is flat. A server name is unique, for all computers on the network. To discover server names, the RTSMB Client can send out a query to the network for all workgroup ‘owners’ to tell it the name of a server in their workgroup that will provide the names of all the servers in that workgroup. The RTSMB Client then queries those servers and builds up its list.

When connecting to a server, you need to first know the server’s name. Then, you establish a session, logon a user, and then connect to a share. All these things must be done before you can browse their files. Establishing a session is easy. Call **rtmsb_cli_session_new_with_name()**. Connecting to a share is also simple. You can query the server for a list of shares, pick one, and then connect to it with an optional password.

Now that you are connected to a server and share, you can use the API calls that mirror standard file I/O routines like open, close, read, write, seek, truncate, etc.

A lot of these details can be dealt with automatically by using the ‘EZ’ API for the RTSMB Client. This API is very simple to use, but only uses blocking mode. So, each call returns only when the task is done. However, the advantage is a clean and simple abstraction of server connections. You simply say, I want to open the file “//smbserver/sharename/path/to/file.txt”. It handles finding the server, logging on a default user, connecting to the share, and then opening the file.

You can also use the EZ API to make searching for servers or shares on a server very simple. For example, you can call the EZ gfirst function with the file pattern “//*server” to get all the servers whose names end with the string “server” or call “//smbserver/*” to get all the shares on the server ‘smbserver’. The EZ API makes the network name space look exactly like a local file system name space.

INTRODUCTION TO THE SMB PROTOCOL

BASICS

This package implements an SMB client. SMB stands for Server Message Block and is a protocol designed by Microsoft, IBM, and others to allow networked computers to share files.

Microsoft uses SMB to allow Windows machines to make resources such as disk trees (SMB vernacular for directories) and printers available to others on the local network.

PROTOCOL HIERARCHY

SMB needs some way of associating network names with addresses. It commonly uses NETBIOS for this, and RTSMB does the same. RTSMB uses TCP/IP for basic network communication and thus the protocols will look like this:

SMB
NETBIOS
IP

BASIC NETWORKING

There are three major ports of interest: 137, 138, and 139.

137 Name Service port (UDP)
138 Netbios Datagram Service port (UDP)
139 Session Service port (TCP)

The name service handles registering of network names and name challenges.

The netbios datagram service handles announcements of server availability.

The session service handles establishing a connection between servers.

The client does not use port 139, since it does not receive session connections. It does, however, use 137 to perform name queries and 138 to ask questions of the network.

Every time the client makes a connection, it connects to the server’s port 139 and opens a new port and assigns that session to it. All further inter-computer communications occurs on that TCP/IP port.

FLOW OF SMB COMMUNICATION

A client will connect to a server by initiating a session request on the servers Session Service port. Then, the server waits until the client issues a request, processes each request, and then issues a response, if needed.

EXAMPLE

Client ————->>>——————— Server
(Connection to port 139; “I want a session”)
Client ————->>>——————— Server
(New connection opened)
.
.
.
Client ————->>>——————— Server
(“open this file”)
Client ————<<<——————— Server
(“success; this is the file id of file”)

The server never initiates a session or a command within a session. Most SMB requests take exactly one response. To start a connection, the client sends a “negotiate” request and receives a response indicating which dialect of SMB will be spoken, as well some server-specific information. Then, the client requests a “session setup” which will log a user in. Finally, to get any work done, that user connects to a share by sending a “tree connect” request. This will allow the user to browse the files on the share (or in the case of IPC$, let the user see what shares are available).

MODULES

The only file that you need to include to have access to the RTSMB Client is cliapi.h, but here is a description of all RTSMB source files:

Common Headers:

smb.h	-	Protocol information
smbobjs.h	-	Protocol data structures
smbconf.h	-	Configuration settings (see below)
smbdefs.h	-	Common defines needed by RTSMB

Porting Modules:

psmbnet	-	Access to the network stack
psmbos	-	Access to OS-level items like mutexes and time

Modules of Client Code:

clians	-	Code to read server responses
cliapi	-	API for the RTSMB Client
clicfg	-	Memory-resident data and configuration
clicmds	-	Code to pack client commands
cliez	-	The control code for the EZ layer
clissn	-	Code to handle a client session
cliwire	-	Code to abstract a connection to a server
smbnb	-	Generic netbios code
smbnbds	-	Netbios datagram service
smbnbns	-	Netbios name service
smbnbss	-	Netbios session service
smbnet	-	Low-level networking code
smbpack	-	Portable packing of data into a buffer
smbread	-	Portable reading of data from a buffer
smbutil	-	Miscellaneous utility functions

PORTING

If you want to port RTSMB Client to a different platform than RTIP/RT-Kernel, Windows, or GNU/Linux, you need to provide two files: psmbnet.c and psmbos.c.

psmbnet.c is an abstraction layer for your network stack. This emulates unix-style sockets. Look at windows/psmbnet.c for examples.

psmbos.c is an abstraction layer for various kernel functions like mutex allocation and timing methods.

To build with your files, simply add them from the correct subdirectory to your project so that they get compiled along with the rest of the source code.

Instructions for porting using Visual C++:

An example project file is under the toplevel directory “RTSMB Client.” If you want to make a new project configuration for your target, take all the source code (.c) files in the toplevel src directory, except for ncbc_enc.c, and add them to your project. Then, add one of both psmbnet.c and psmbos.c from whichever subdirectory is appropriate (or the ones you made).

RT-SMB does not require any command line defines or special link libraries. Just link the libraries you need for your net and os files and add the following preprocessor includes (under Project->Settings and then the C/C++ tab, category Preprocessor, field “Additional include directories:”):

..\src\include,..\src\include\OPENSSL,..\src

RTIP 4.0

SMB Client Manual

Chapter Two: Configuring SMB

CHAPTER 2

PARAMETERS LOCATED IN CLICFG.C.

CFG_RTSMB_CLI_MAX_SESSIONS
The maximum number of simultaneous sessions that can exist. This is synonymous with the maximum number of servers we can connect to at once. Very often, setting this to 1 is fine. These sessions are quite large; be careful when setting this high. ***The default is 1.***

CFG_RTSMB_CLI_MAX_SEARCHES_PER_SESSION
The number of file searches that can exist at once on a session. It is unlikely that you would need this much higher than 1. Cannot be over 255. ***The default is 2.***

CFG_RTSMB_CLI_MAX_FILES_PER_SEARCH
The number of files that can be returned in one search iteration. Increasing this improves network efficiency at the expense of more memory use. ***The default is 10.***

CFG_RTSMB_CLI_MAX_FIDS_PER_SESSION
The number of open files that can exist at once on a session. Cannot be over 255. ***The default is 10.***

CFG_RTSMB_CLI_MAX_SHARES_PER_SESSION
The number of shares that a client can be connected to at once. This includes the IPC share which a client is always connected to. ***The default is 4.***

CFG_RTSMB_CLI_MAX_SHARES_PER_SEARCH
The number of shares that can be returned in one share enumeration. Increasing this improves network efficiency at the expense of more memory use. ***The default is 10.***

CFG_RTSMB_CLI_MAX_SERVER_SEARCHES
The number of servers enumerations that can simultaneously exist. This is unlikely to be wanted much larger than 1. Do not make this larger than 255. ***The default is 1.***

CFG_RTSMB_CLI_MAX_SERVERS_PER_SEARCH
This controls how many servers one search can return at once. This means that this controls how many servers you will find from each workgroup on the network. Since each server only takes up 16 bytes of information, it seems low-risk to leave this high. ***The default is 50.***

CFG_RTSMB_CLI_BUFFER_SIZE
This controls how much data can be sent per packet. Lower values decrease network performance due to more packets sent and the overhead they cost (only a concern if making reads or writes larger than the packet size). Higher values use more memory. Pick your poison. Must be at least 1028. Windows 95 uses around 2k. Windows XP uses around 4k. ***The default is 1028.***

CFG_RTSMB_CLI_MAX_BUFFERS_PER_WIRE
This controls how many buffers we have per wire. A wire is essentially a connection to a server. If we have more buffers, we can do more simultaneous activities on that connection. This only needs to be non-1 if you are using the **raw rtsmb_cli_session_*** API and want to do asynchronous jobs. Otherwise, just set this to 1 and save some memory (each buffer takes up **CFG_RTSMB_CLI_BUFFER_SIZE** bytes). ***The default is 1.***

CFG_RTSMB_CLI_MAX_JOBS_PER_SESSION
This controls how many jobs a session is allowed to have outstanding. Since each job uses one buffer in the wire, there is not much reason to have this different than the number of buffers per wire. However, there should be one extra job allotted for internal use. ***The default is CFG_RTSMB_CLI_MAX_BUFFERS_PER_WIRE + 1.***

CFG_RTSMB_CLI_MAX_SUPPORTED_THREADS
This controls how many different threads the EZ API can be run on with no problems. Right now, this is only used to control how many different ‘current working directories’ we keep track of (one per thread). Set this to however many threads you expect to use the EZ API from. ***The default is 1***.

RTIP 4.0

SMB CLIENT MANUAL

CHAPTER THREE: PORTING SMB

CHAPTER 3

RTSMB PORTING: OVERVIEW

The interface from RTSMB to the underlying operating system, network stack, and file system is provided through the following files:

psmbos.c - interface to underlying OS, including mutex semaphores, time functions, and printer functions (if printer support is desired)

psmbnet.c - interface to TCP/IP network stack; modeled on the BSD sockets interface

This section describes the functions contained within these files, and how to port RTSMB to an environment.

If you want to port RT-SMB to a new platform, you need to provide the three porting files – psmbfile.c, psmbnet.c, and psmbos.c.

To build with your files, simply add them from the correct subdirectory to your project so that they get compiled along with the rest of the source code.

INSTRUCTIONS FOR PORTING USING VISUAL C++:

An example project file is under the toplevel directory "RTSMB Server."

If you want to make a new project configuration for your target, take all the source code (.c) files in the toplevel src directory, except for ncbc_enc.c, and add them to your project. Then, add one of each of psmbfile.c, psmbnet.c, and psmbos.c from whichever subdirectory is appropriate (or the ones you made).

RT-SMB does not require any command line defines or special link libraries. Just link the libraries you need for your net, OS, and file porting files and add the following preprocessor includes (under Project->Settings and then the C/C++ tab, category Preprocessor, field "Additional include directories:"):

..\src\include,..\src\include\OPENSSL,..\src

RTSMB PORTING: PERIODIC CLOCK SUPPORT (PSMBOS.C)

RTSMB requires a periodic clock for protocol-related timeouts. A single function provides the interface to this service:

unsigned long **rtsmb_osport_get_msec** (void)

Returns the system clock time in milliseconds.

RTSMB Porting: Mutex Semaphore Support (psmbos.c)

Note: Mutex support is only required by RTSMB in a multi-tasking environment. If RTSMB is invoked from a single thread in polled mode, mutex semaphore support is not necessary.

FUNCTIONS NECESSARY TO PROVIDE MUTEX SEMAPHORE SUPPORT FOR RTSMB:

int **rtsmb_osport_create_mutex** (unsigned long *mutexHandle)

This routine must allocate and initialize a mutex, to the unclaimed status. It mustset the unsigned long pointed to by mutexHandle to a value that can be used as a handle to the mutex. If this routine is successful, this routine returns 0. Otherwise, it returns a negative value and the value of *mutexHandle is undefined.

void **rtsmb_osport_claim_mutex** (unsigned long mutexHandle)

This routine takes a mutex handle returned by rtsmb_osport_create_mutex and returns nothing. If the mutex is already claimed, this routine must wait for the mutex to be released and then claim it and return.

void **rtsmb_osport_release_mutex** (unsigned long mutexHandle)

This routine takes a mutex handle returned by rtsmb_osport_create_mutex and releases it. It returns nothing.

RTSMB PORTING: THREAD SUPPORT (PSMBOS.C)

These functions are only required if RTSMB is to be run in multi-threaded mode.

int **rtsmb_osport_create_thread** (RTSMB_THREAD_FN fn, void *context)

This routine takes two parameters, a thread entry point function and a context pointer to pass into this entry point. It returns 0 if the thread is successfully spawned, a negative value otherwise. The entry point function takes a single value, a void pointer, and returns nothing. This routine should create a thread and start it running, starting at the given function. The context parameter should be passed into the entry point function.

void **rtsmb_osport_exit_thread** (void)

This routine should perform any kernel specific thread termination and clean up operations. It will be the last thing done in any thread created by RTSMB using rtsmb_osport_create_thread. It signals the termination of the thread from which it is called.

RTSMB PORTING: NETWORK SUPPORT (PSMBNET.C)

The RTSMB interface to the underlying TCP/IP network stack is modeled after the BSD sockets API, but with some modifications to eliminate dependencies on platform-specific types and structures. As much as possible, the intention is to provide a very direct mapping between RTSMB network porting functions and calls in the sockets API. Referring to a document describing the sockets API, such as the RTIP manual, may therefore be helpful in porting psmbnet to a particular network stack.

FUNCTIONS THAT MUST BE PROVIDED AS AN INTERFACE TO THE NETWORK STACK

Note: all network addresses are arrays of 4 unsigned chars. All port values are integers in host byte order (port values may need therefore to be converted to network byte order within these routines).

int **rtsmb_netport_init** (void)

This function should initialize the network stack for operation. If successful, it returns 0, otherwise, it returns a negative value.

int **rtsmb_netport_accept**

(int *accepted, int socketId, unsigned char *remoteAddr, int *remotePort)

accepted	-	pointer to a socket handle to set to the socket id of the new connection
socketId	-	the socket to accept the connection on
remoteAddr	-	(optional) array to fill with the ip address of the remote host
remotePort	-	(optional) pointer to int to set to the port of the remote host

Description:

This function should block waiting for a remote host to connect to the port/ip address to which socketId is bound. If a connection is successfully established, it must set *newConnection to the socket for the new connection (socketId continues to listen on its port/ip), and return 0.

In the event of a successful connection, remoteAddr and remotePort should also be set to the ip/port of the remote host that connected. If no connection can be established or an error occurs, the return value is negative, and the values of *newConnection, *remoteAddr, and *remotePort are undefined.

The behavior of this function is undefined in the following cases:

- socketId is not a valid, stream-type socket
- rtsmb_netport_bind was never called on socketId
- rtsmb_netport_listen was never called on socketId

See Also:

rtsmb_netport_bind, rtsmb_netport_listen,
rtsmb_netport_socket_stream

Returns:

0 on success, negative on failure

int **rtsmb_netport_allow_broadcast**

(int socketId)

socketId - the socket for broadcasting

Description:

This function must be called on any socket over which broadcast messages are to be sent. If it is not called, broadcast messages are not guaranteed to work.

Returns:

0 on success, negative on failure

int **rtsmb_netport_bind**

(int socketId, unsigned char *ipAddr, int port)

socketId	-	the socket to bind
ipAddr	-	(optional) the ip address to associate with this socket
port	-	the port to associate with this socket

Description:

This function should be called on a socket before calling rtsmb_netport_listen/rtsmb_netport_accept (if the socket is stream-type), or rtsmb_netport_recvfrom (if the socket is datagram-type). The behavior of this function is undefined if socketId is not a valid socket handle.

If the port specified is already in use by another socket, this function must return a negative value.

See Also:

rtsmb_netport_accept, rtsmb_netport_listen,
rtsmb_netport_socket_stream

Returns:

0 on success, negative on failure

int **rtsmb_netport_closesocket**

(int socketId)

socketId - the socket to close

Description:

This function should be called to release a socket and shut down any open connection it has. It is not defined whether this is a hard close.

See Also:

rtsmb_netport_accept, rtsmb_netport_connect

Returns:

0 on success, negative on failure

int **rtsmb_netport_connect**

(int socketId, unsigned char *ipAddr, int port)

socketId	-	the socket to connect
ipAddr	-	the address to connect to
port	-	the port to connect to

Description:

This function is called on a stream-type socket to initiate a connection to a specific ip address and port. This function can block until the connection is established. Behavior is undefined if socketId is not a valid, stream-type socket handle.

See Also:

rtsmb_netport_accept, rtsmb_netport_socket_stream, rtsmb_netport_closesocket

Returns:

0 on success, negative on failure

int **rtsmb_netport_listen**

(int socketId, int queueSize)

socketId	-	the socket handle
queueSize	-	the max number of requested connections to queue for acceptance

Description:

This function is called on a stream-type socket after rtsmb_netport_bind but before rtsmb_netport_accept to get a socket ready to accept connections from remote hosts.

See Also:

rtsmb_netport_accept, rtsmb_netport_bind, rtsmb_netport_socket_stream

Returns:

0 on success, negative on failure

long **rtsmb_netport_recv**

(int socketId, unsigned char *buffer, long size)

socketId	-	the socket to read from
buffer	-	pointer to a buffer to place received data in
size	-	the maximum number of bytes to read

Description:

This function is called on a connected socket to read data off that socket. It can block until up to size bytes are read or until the connection is closed, signifying that no more bytes are coming. If the connection has been terminated, it can return either 0 or a negative value.

If socketId is not a valid handle to a stream-type socket, behavior is undefined.

See Also:

rtsmb_netport_recvfrom, rtsmb_netport_send, rtsmb_netport_sendto, rtsmb_netport_select_n_for_read

Returns:

number of bytes read on success, negative on failure

long rtsmb_netport_recvfrom

(int socketId, unsigned char *buffer, long size, unsigned char *ipAddr, int *port)

socketId	-	the socket to read from
buffer	-	pointer to a buffer to place received data in
size	-	the maximum number of bytes to read
ipAddr	-	(optional) pointer to a 4-byte array to fill with the ip address of the sender
port	-	(optional) pointer to an int to set to the port of the sending socket

Description:

This function is called on a connectionless socket to read data. This function should block until data is ready, then read at most size bytes into buffer. If the datagram is smaller than size bytes, then only the number of bytes in the datagram must be read into the buffer, and the function must return.

See Also:

rtsmb_netport_recv, rtsmb_netport_send, rtsmb_netport_sendto, rtsmb_netport_select_n_for_read

Returns:

number of bytes read on success, negative on failure

int **rtsmb_netport_select_n_for_read**

(int *socketList, int listSize, long timeoutMsec)

socketList	-	an array of socket ids to select from
listSize	-	the number of elements in socketList
timeoutMsec	-	the maximum number of milliseconds to wait before timing out. negative value means wait forever.

Description:

This function must block for at most timeoutMsec milliseconds (or forever if this value is negative) waiting for data to become available for reading on at least one of the sockets listed in the socketList array.

When this function returns, it must have modified the contents of socketList to contain only those sockets which are ready for reading. The return value is the number of sockets on this modified list (i.e. the number of sockets that have data ready to be read).

See Also:

rtsmb_netport_recv, rtsmb_netport_recvfrom

Returns:

the number of sockets that have data ready to read.

long **rtsmb_netport_send**

(int socketId, unsigned char *buffer, long size)

socketId	-	the socket to send over
buffer	-	pointer to data to send
size	-	the number of bytes to send

Description:

This function is used to send data over a connected socket. The socket must be a stream-type; behavior is undefined if socketId is a datagram-type (connectionless) socket.

See Also:

rtsmb_netport_sendto, rtsmb_netport_recv, rtsmb_netport_recvfrom

Returns:

The number of bytes sent if successful, negative if an error occurred

long **rtsmb_netport_sendto**

(int socketId, unsigned char *buffer, long size, unsigned char *ipAddr, int port)

socketId	-	the socket to send over
buffer	-	pointer to data to send
size	-	the number of bytes to send
ipAddr	-	the IP address to send to
port	-	the port number to send to

Description:

Sends size bytes from buffer to the specified ip address/ port. The given socket must be a connectionless (datagram-type); otherwise, behavior is undefined.

See Also:

rtsmb_netport_send, rtsmb_netport_recv, rtsmb_netport_recvfrom

Returns:

Returns the number of bytes sent if successful, negative if an error occurred

int **rtsmb_netport_socket_stream**

(int *socketId)

socketId - pointer to an int to set to the socket handler

Description:

Allocates, if possible, a stream (TCP, connection-based) type socket. If return value is negative, the value of *socketId is undefined.

Returns:

0 if successful, negative otherwise

int **rtsmb_netport_socket_datagram**

(int *socketId)

socketId - pointer to an int to set to the socket handler

Description:

Allocates, if possible, a datagram (UDP, connectionless) type socket. If return value is negative, the value of *socketId is undefined.

Returns:

0 if successful, negative otherwise

RTIP 4.0

SMB Client Manual

Chapter Four: SMB Client API

CHAPTER 4

API FUNCTIONS

The following pages document each call to the RTSMB Client API. But first, a quick categorical overview of the functions.

GLOBAL CLIENT MANAGEMENT:

rtsmb_cli_init

This initializes global client data and should be called before anything else.

rtsmb_cli_shutdown

This frees global client data and should be the last function called.

SESSION MANAGEMENT:

rtsmb_cli_session_new_with_name

This initializes a session and begins the connection to a server.

rtsmb_cli_session_close

This closes a session.

rtsmb_cli_session_restart

This closes and then reopens a session.

TASK SYNCHRONIZATION:

rtsmb_cli_session_set_blocking

This sets whether or not session calls should block until the task is complete.

rtsmb_cli_session_cycle

This, if a session is in non-blocking mode, will allow the session to do its work, optionally blocking for a timeout.

rtsmb_cli_session_set_job_callback

This, if a session is in non-blocking mode, will set a function to be called when a task completes.

CONNECTION MANAGEMENT:

rtsmb_cli_session_logon_user

This logs on a user to the server, which may be necessary before it lets you connect to a share. Only one user can be logged on at once.

rtsmb_cli_session_logoff_user

This logs the user off.

rtsmb_cli_session_connect_share

This connects the session to a server share for future reads or writes.

rtsmb_cli_session_disconnect_share

This disconnects the session to a server share.

rtsmb_cli_session_share_find_first

Begins a query to the server to enumerate its shares.

rtsmb_cli_session_share_find_next

Gets the next share name in the list of shares available.

rtsmb_cli_session_share_find_close

Ends a query to the server to enumerate its shares.

File I/O:

rtsmb_cli_session_open
rtsmb_cli_session_close
rtsmb_cli_session_read
rtsmb_cli_session_write
rtsmb_cli_session_seek
rtsmb_cli_session_truncate
rtsmb_cli_session_flush
rtsmb_cli_session_rename
rtsmb_cli_session_delete
rtsmb_cli_session_mkdir
rtsmb_cli_session_rmdir
rtsmb_cli_session_find_first
rtsmb_cli_session_find_next
rtsmb_cli_session_find_close
rtsmb_cli_session_stat
rtsmb_cli_session_chmode
rtsmb_cli_session_get_free

EZ File I/O:

rtsmb_cli_ez_set_user

This sets which user to logon to a server as. You don't need to call this if you don't have a need.

rtsmb_cli_ez_open
rtsmb_cli_ez_read
rtsmb_cli_ez_write
rtsmb_cli_ez_seek
rtsmb_cli_ez_close
rtsmb_cli_ez_truncate
rtsmb_cli_ez_flush
rtsmb_cli_ez_rename
rtsmb_cli_ez_delete
rtsmb_cli_ez_mkdir
rtsmb_cli_ez_rmdir
rtsmb_cli_ez_find_first
rtsmb_cli_ez_find_next
rtsmb_cli_ez_find_close
rtsmb_cli_ez_stat
rtsmb_cli_ez_chmode
rtsmb_cli_ez_get_cwd
rtsmb_cli_ez_set_cwd
rtsmb_cli_ez_get_free

Any function that takes a filename has a complement function with the same name except a suffix of "_uc" to indicate it takes a Unicode string instead of an ASCII string. These functions only exist if RTSMB was compiled with Unicode support.

RTSMB_CLI_INIT

Function:

Initialize global data.

Summary:

#include “cliapi.h”

int **rtsmb_cli_init** (ip, mask_ip)

PFBYTE ip	- the four-byte host ip to use
PFBYTE mask_ip	- the four-byte subnet mask to use

Description:

This should be called before any other RTSMB Client function. Pass in two four-byte arrays representing an ip and a subnet mask. If subnet mask is NULL, the default subnet of 255.255.255.0 is used.

Returns:

Returns zero on success or a negative value on error

RTSMB_CLI_SHUTDOWN

Function:

Free global data.

Summary:

#include “cliapi.h”

void **rtsmb_cli_shutdown** ()

Description:

This should be the last client function called. It frees up any global resources used by the client. It also closes all currently open client sessions.

RTSMB_CLI_SESSION_NEW_WITH_NAME

Function:

Initialize a session.

Summary:

#include "cliapi.h"

int **rtsmb_cli_session_new_with_name** (name, blocking, broadcast_ip, psid)

PFCHAR name	- the name of the server with which to connect
BBOOL blocking	- whether this session can block or not
PFBYTE broadcast_ip	- the four-byte broadcast ip to use
PFINT psid	- an int to fill with the session id of the new session

Description:

This function finds a free session, initializes it, and connects to the server. NULL can be passed for the broadcast ip and the global default will be used.

Returns:

In blocking mode, returns a zero on success or a negative value on failure. In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_CLOSE

Function:

Close a session.

Summary:

#include "cliapi.h"

void **rtsmb_cli_session_close** (sid)

int sid - the session id you wish to close

Description:

This function shuts down the session you specify.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_RESTART

Function:

Restart a session.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_restart** (sid)

int sid - the session id you wish to restart

Description:

This function restarts the session you specify. The connection to the server is torn down, and created again. All logged on users, connected shares, and open files are reestablished. You shouldn’t need to use this on a regular basis. Helpful if you feel something is wrong with the connection.

Returns:

Returns a zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_SET_BLOCKING

Function:

Sets a session to blocking or non-blocking mode.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_set_blocking** (sid, blocking)

int sid	-	the session id you wish to set
BBOOL blocking	-	whether or not to block

Description:

This function sets the session to blocking mode if ‘blocking’ is non-zero and to non-blocking mode if ‘blocking’ is zero. In blocking mode, each call will not return until the task is complete. In non-blocking mode, each call will return without blocking on the network socket and will return a job id.

Returns:

Returns a zero on success or a negative number on failure (bad session id).

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_CYCLE

Function:

Let a session work.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_cycle** (sid, timeout)

int sid	- the session id you wish to cycle
long timeout	- the maximum number of milliseconds to block

Description:

This function is only needed when in non-blocking mode. It will check the network socket and process incoming answers, or it will send off waiting requests. It will block no more than ‘timeout’ milliseconds.

Returns:

Returns a zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_SET_JOB_CALLBACK

Function:

Set a callback for when a job is complete.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_set_job_callback** (sid, job, callback_func, callback_data)

int sid - the session id you wish to use

int job - the job id you wish to use

RTSMB_JOB_CALLBACK callback_func-the function to call when the job is done

PFVOID callback_data - data to pass to the callback function

Description:

This function is only needed when in non-blocking mode. It will set the callback function to be called upon completion of the job. This means upon successful completion, upon an error, or upon a timeout. The callback function is a void function taking an int job id, an int job return value, and a PFVOID ‘data’ value.

The reason you would set a job callback is that when you are running in non-blocking mode, there is no other way to tell when a task you started has been responded to. You cannot assume that any return parameters are valid until the callback has been called. At that point, the task has been completed.

Returns:

Returns a zero on success or a negative value on failure (session, job not found).

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_LOGON_USER

Function:

Logon a user.

#include “cliapi.h”

int **rtsmb_cli_session_logon_user** (sid, user, password)

int sid	-	the session id you wish to use
PFCHAR user	-	the username to use
PFCHAR password	-	the password to use

Description:

This function attempts to logon a user to the server. Without logging on a user, servers that require user logons will not let you connect to a share. So, for every server, you should always try to log on some user — if they don’t recognize you, they often will let you in with guest privileges. Only one user can be logged on at once.

Returns:

In blocking mode, returns a zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_LOGOFF_USER

Function:

Logoff a user.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_logoff_user** (sid, user)

int sid	-	the session id you wish to use
PFCHAR user	-	the username to logoff

Description:

This function logs a user off the server.

Returns:

In blocking mode, returns a zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_CONNECT_SHARE

Function:

Connect to a shared directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_connect_share** (sid, share, password)

int sid	-	the session id you wish to use
PFCHAR share	-	the share name to use
PFCHAR password	-	the password to use

Description:

This function attempts to connect to a share on the server. Without connecting to a share, you cannot do any file I/O on the share. The password is only used if the server is running in share-level authentication. You can pass NULL for it or the empty string to enter no password.

Returns:

In blocking mode, returns a zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_DISCONNECT_SHARE

Function:

Disconnect from a shared directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_disconnect_share** (sid, share)

int sid	-	the session id you wish to use
PFCHAR share	-	the share name from which to disconnect

Description:

This function attempts to disconnect from a share on the server.

Returns:

In blocking mode, returns a zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_SHARE_FIND_FIRST

Function:

Query the server to enumerate its shares.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_share_find_first** (sid, pstat)

int sid - the session id you wish to use

PRTSMB_CLI_SESSION_SSTAT pstat -a share info struct to fill out

Description:

This function queries the server about what shares are available. It returns a list of share names. The first is returned in pstat->name. Further share names can be discovered by calling **rtsmb_cli_session_share_find_next**.

Returns:

In blocking mode, **rtsmb_cli_session_share_find_first** returns **CSSN_RV_SEARCH_DATA_READY** on availability of a share name, **CSSN_RV_END_OF_SEARCH** when no more names are available, or another negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_SHARE_FIND_NEXT

Function:

Get the next share name in a list.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_share_find_next** (sid, pstat)

int sid - the session id you wish to use

PRTSMB_CLI_SESSION_SSTAT pstat - a share info struct to fill out

Description:

This function returns the next share name in pstat->name. Further share names can be discovered by calling **rtsmb_cli_session_share_find_next** again. This function does not need to wait on the server; data is either ready or the search is over.

Returns:

Returns CSSN_RV_SEARCH_DATA_READY on availability of a share name, CSSN_RV_END_OF_SEARCH when no more names are available, or another negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_SHARE_FIND_CLOSE

Function:

End a share name query.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_share_find_close** (sid, pstat)

int sid - the session id you wish to use

PRTSMB_CLI_SESSION_SSTAT pstat - a share info struct to fill out

Description:

This function ends the share name query and frees resources for future share name queries.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_OPEN

Function:

Open a file for reading or writing.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_open** (sid, share, filename, flags, mode, pfid)

int sid	- the session id you wish to use
PFCHAR share	- the share name to use
PFCHAR filename	- the filename to open
int flags	- the flags to open the file with
int mode	- the mode to create the file with (if necessary)
PFINT pfid	- an integer to fill with the file id when the job is done

Description:

This function opens the specified filename.

‘Flags’ can be a bitmask of the following values:

RTSMB_O_RDONLY	- (read only priviledges)
RTSMB_O_WRONLY	- (write only priviledges)
RTSMB_O_RDWR	- (both read and write priviledges)
RTSMB_O_CREAT	- (create the file if it doesn’t exist)
RTSMB_O_EXCL	- (when creating, fail if the file exists)
RTSMB_O_TRUNC	- (truncate the file to 0 bytes after opening)

‘Mode’ can be a bitmask of the following values:

RTSMB_S_IWRITE
RTSMB_S_IREAD

‘Mode’ indicates what permissions the file is created with, if it is created. It has no meaning if RTSMB_O_CREAT isn’t specified.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_CLOSE

Function:

Close a file.

Summary:

#include "cliapi.h"

int **rtsmb_cli_session_close** (sid, fid)

int sid	- the session id you wish to use
int fid	- the file id to close

Description:

This function closes the specified file.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_READ

Function:

Read data from a file.

Summary:

#include "cliapi.h"

int **rtsmb_cli_session_read** (sid, fid, pdata, count, pcount_read)

int sid	- the session id you wish to use
int fid	- the file id to use
PFBYTE pdata	- a buffer to fill with data
int count	- a maximum number of bytes to read
PFINT pcount_read	- a word to fill with the number of bytes read when job is done

Description:

This function reads data from the file. No more than 'count' bytes will be read. If less bytes are read, the end of the file was reached.

The value of 'count' cannot exceed the value of **RTSMB_CLI_SESSION_MAX_DATA_BYTES**. This is the maximum number of bytes that can be read in one read packet. An attempt to read more will result in an error.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_WRITE

Function:

Write data to a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_write** (sid, fid, pdata, count, pwritten)

int sid	- the session id you wish to use
int fid	- the file id to use
PFBYTE pdata	- a buffer from which data is read
int count	- the number of bytes to write
PFINT pwritten	- a word to fill with the number of bytes written when job is done

Description:

This function writes data to the file.

The value of ‘count’ cannot exceed the value of **RTSMB_CLI_SESSION_MAX_DATA_BYTES**. This is the maximum number of bytes that can be sent in one write packet. An attempt to send more will result in an error.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_SEEK

Function:

Seek to an offset in a file.

Summary:

#include “cliapi.h”

long **rtsmb_cli_session_seek** (sid, fid, offset, location, presulting_offset)

int sid	- the session id you wish to use
int fid	- the file id to use
long offset	- how many bytes to offset
int location	- the starting location for the offset
PFLONG presulting_offset	- a dword to fill with the resulting offset from start of file

Description:

This function sets the current offset in the file. ‘Location’ may be RTSMB_SEEK_SET (from the beginning of the file), RTSMB_SEEK_CUR (from the current location in the file), or RTSMB_SEEK_END (from the end of the file).

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_TRUNCATE

Function:

Change the size of a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_truncate** (sid, fid, offset)

int sid	-	the session id you wish to use
int fid	-	the file id to use
long offset	-	the new size of the file

Description:

This function sets the size of the file. It can destroy data past the offset or extend the file to the offset.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_FLUSH

Function:

Flush a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_flush** (sid, fid)

int sid	-	the session id you wish to use
int fid	-	the file id to flush

Description:

This function flushes a file. All waiting I/O requests will be complete on the server before it sends a response.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_RENAME

Function:

Rename a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_rename** (sid, share, old_filename, new_filename)

int sid	-	the session id you wish to use
PFCHAR share	-	the share name to use
PFCHAR old_filename	-	the current filename
PFCHAR new_filename	-	the desired new filename

Description:

This function renames a file. Both must be on the same share and the new filename must not exist.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_DELETE

Function:

Delete a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_delete** (sid, share, filename)

int sid	-	the session id you wish to use
PFCHAR share	-	the share name to use
PFCHAR filename	-	the filename to delete

Description:

This function deletes a file.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_MKDIR

Function:

Make a directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_mkdir** (sid, share, filename)

int sid	-	the session id you wish to use
PFCHAR share	-	the share name to use
PFCHAR filename	-	the directory name to create

Description:

This function creates a directory at the specified path.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_RMDIR

Function:

Delete a directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_rmdir** (sid, share, filename)

int sid	-	the session id you wish to use
PFCHAR share	-	the share name to use
PFCHAR filename	-	the directory name to delete

Description:

This function deletes a directory at the specified path.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_FIND_FIRST

Function:

Start a directory traversal.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_find_first** (sid, share, pattern, pdstat)

int sid	- the session id you wish to use
PFCHAR share	- the share name to use
PFCHAR pattern	- the filename pattern to use
PRTSMB_CLI_SESSION_DSTAT pdstat	- the stat structure to fill out with information

Description:

This function begins a directory traversal, searching for all files that match the pattern specified. The struct pdstat will be filled out with data from the first matching file, if any.

Returns:

In blocking mode, **RTSMB_CLI_SESSION_FIND_FIRST** returns **CSSN_RV_SEARCH_DATA_READY** on availability of a file’s information, **CSSN_RV_END_OF_SEARCH** when no more files are available, or another negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_FIND_NEXT

Function:

Continue a directory traversal.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_find_next** (sid, pdstat)

int sid	- the session id you wish to use
PRTSMB_CLI_SESSION_DSTAT pdstat	- the stat structure to fill out with information

Description:

This function continues a directory traversal, searching for all files that match the original pattern specified. The struct pdstat will be filled out with data from the next matching file, if any.

Returns:

In blocking mode, **RTSMB_CLI_SESSION_FIND_NEXT** returns **CSSN_RV_SEARCH_DATA_READY** on availability of a file’s information, **CSSN_RV_END_OF_SEARCH** when no more files are available, or another negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_FIND_CLOSE

Function:

End a directory traversal.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_find_close** (sid, pdstat)

int sid - the session id you wish to use

PRTSMB_CLI_SESSION_DSTAT pdstat - the stat structure used in prior calls

Description:

This function ends a directory traversal, freeing up the resources for future file searches.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_STAT

Function:

Discover information about a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_stat** (sid, share, filename, pfstat)

int sid	- the session id you wish to use
PFCHAR share	- the share name to use
PFCHAR filename	- the filename to query

PRTSMB_CLI_SESSION_FSTAT pfstat - the stat structure to fill with data

Description:

This function queries for information about a particular file. Upon completion, pfstat contains the information.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_CHMODE

Function:

Change file permissions.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_chmode** (sid, share, filename, attributes)

int sid	- the session id you wish to use
PFCHAR share	- the share name to use
PFCHAR filename	- the filename to change
int attributes	- the new attributes of the file

Description:

This function changes the attributes of a particular file. Attributes is a bitmask with one of the following values set: **RTSMB_O_RDONLY**, **RTSMB_O_WRONLY**, or **RTSMB_O_RDWR**.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_SESSION_GET_FREE

Function:

Query disk size.

Summary:

#include “cliapi.h”

int **rtsmb_cli_session_get_free** (sid, share, ptotal_units, pfree_units, psectors_per_unit, pbytes_per_sector)

int sid	- the session id you wish to use
PFCHAR share	- the share name to check size of
PFINT ptotal_units	- return location for how many total units are on the disk
PFINT pfree_units	- return location for how many free units are on the disk
PFINT psectors_per_unit	- return location for how many sectors exist in each unit
PFINT pbytes_per_sector	- return location for how many bytes each sector takes up

Description:

This function queries the size of the disk on which the specified share resides.

Returns:

In blocking mode, returns zero on success or a negative value on failure.

In non-blocking mode, returns a non-negative value on success indicating the job id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

EZ API

A word about how the EZ layer works. All functions block until the task is complete. Some functions take a URI (Universal Resource Identifier) parameter, which is a string representing an SMB resource. These URI's takes the following form: "//server/share/path/to/file.txt". The forward slashes may also be backward slashes, but do not mix them in the same URI. In addition, the URI may be prefixed by the string "smb:". Some functions may allow partial URI's, like "//server/share" or even "//server".

RTSMB_CLI_EZ_SET_USER

Function:

Set the default user for EZ sessions.

Summary:

#include "cliapi.h"

void **rtsmb_cli_ez_set_user** (name, password)

PFCHAR name	-	the username to use
PFCHAR password	-	the password to use

Description:

This function sets the default username and password that the EZ layer will use when creating sessions or connecting to shares. If NULL is passed or this is never called, the internal defaults are used of 'anonymous' for the username and no password.

RTSMB_CLI_EZ_OPEN

Function:

Open a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_open** (uri, flags, mode)

PFCHAR uri	- the URI of the file you wish to open
int flags	- the flags with which to open the file
int mode	- the mode in which to create the file (if necessary)

Description:

This function opens the specified filename.

‘Flags’ can be a bitmask of the following values:

RTSMB_O_RDONLY	- (read only priviledges)
RTSMB_O_WRONLY	- (write only priviledges)
RTSMB_O_RDWR	- (both read and write priviledges)
RTSMB_O_CREAT	- (create the file if it doesn’t exist)
RTSMB_O_EXCL	- (when creating, fail if the file exists)
RTSMB_O_TRUNC	- (truncate the file to 0 bytes after opening)

‘Mode’ can be a bitmask of the following values:

RTSMB_S_IWRITE
RTSMB_S_IREAD

‘Mode’ indicates what permissions the file is created with, if it is created. It has no meaning if RTSMB_O_CREAT isn’t specified.

Returns:

Returns a non-negative value on success indicating the file id or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_READ

Function:

Read from a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_read** (fid, buffer, count)

int fid	- the file id to use
PFBYTE buffer	- the buffer to fill with data
unsigned int count	- the maximum number of bytes to read

Description:

This function reads from a file. If less than ‘count’ bytes are read, the end of the file was reached.

Returns:

Returns a non-negative number of bytes read on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_WRITE

Function:

Write to a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_write** (fid, buffer, count)

int fid	- the file id to use
PFBYTE buffer	- the buffer from which to read data
unsigned int count	- the maximum number of bytes to write

Description:

This function writes to a file.

Returns:

Returns a non-negative number of bytes written on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_SEEK

Function:

Seek to an offset in a file.

Summary:

#include “cliapi.h”

long **rtsmb_cli_ez_seek** (fid, offset, location)

int fid	- the file id to use
long offset	- the number of bytes to offset
int location	- the starting location for the offset

Description:

This function seeks to a new offset in the file. ‘Location’ may be either **RTSMB_SEEK_SET** (beginning of the file), **RTSMB_SEEK_CUR** (current location in the file), or **RTSMB_SEEK_END** (end of the file).

Returns:

Returns a non-negative number on success indicating the new offset from the start of the file or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_CLOSE

Function:

Close a file.

Summary:

#include "cliapi.h"

int **rtsmb_cli_ez_close** (fid)

int fid - the file id to close

Description:

This function closes a file.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_TRUNCATE

Function:

Change the size of a file.

Summary:

#include "cliapi.h"

int **rtsmb_cli_ez_truncate** (fid, offset)

int fid	- the file id to use
long offset	- the new size of the file in bytes

Description:

This function changes the size of a file. It may destroy data past the offset or extend the file to the offset.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_FLUSH

Function:

Flush a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_flush** (fid)

int fid	- the file id to flush

Description:

This function flushes a file. All waiting file I/O on the server is completed before this function returns.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_RENAME

Function:

Rename a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_rename** (old_filename, new_filename)

PFCHAR old_filename	- the current filename to change
PFCHAR new_filename	- the new desired filename

Description:

This function renames a file. The new filename must not exist and both must be on the same server and share.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_DELETE

Function:

Delete a file.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_delete** (filename)

PFCHAR filename - the filename to delete

Description:

This function deletes a file.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_MKDIR

Function:

Create a directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_mkdir** (filename)

PFCHAR filename - the directory to create

Description:

This function creates a directory on the specified server and share.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_RMDIR

Function:

Delete a directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_rmdir** (filename)

PFCHAR filename - the directory to delete

Description:

This function deletes a directory on the specified server and share.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_FIND_FIRST

Function:

Start a directory traversal.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_find_first** (pattern, pdstat)

PFCHAR pattern - the pattern to match

PRTSMB_CLI_SESSION_DSTAT pdstat - the struct to fill out with data

Description:

This function finds the first matching file and fills out ‘pdstat’ with the information from it. This can also accept partial URI’s, such as “//*s” which will return the first server name that ends in ‘s’, or “//smbserver/??” which will return the first share name on ‘smbserver’ that is exactly two characters long.

Returns:

Returns 1 if data is ready in pdstat, 0 if the end of the search has been reached and there is no new data in pdstat, or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_FIND_NEXT

Function:

Continue a directory traversal.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_find_next** (pdstat)

PRTSMB_CLI_SESSION_DSTAT pdstat - the struct to fill out with data

Description:

This function finds the next matching file according to pdstat’s originating pattern and fills out ‘pdstat’ with the information from it.

Returns:

Returns 1 if data is ready in pdstat, 0 if the end of the search has been reached and there is no new data in pdstat, or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_FIND_CLOSE

Function:

End a directory traversal.

Summary:

#include “cliapi.h”

void **rtsmb_cli_ez_find_close** (pdstat)

PRTSMB_CLI_SESSION_DSTAT pdstat - the struct from prior calls

Description:

This function frees the resources associated with the file search.

RTSMB_CLI_EZ_FIND_STAT

Function:

Discover information about a file.

Summary:

#include “cliapi.h”

void **rtsmb_cli_ez_stat** (filename, pfstat)

PFCHAR filename - the filename to stat

PRTSMB_CLI_SESSION_FSTAT pfstat - the struct to fill with data

Description:

This function obtains information about the named file and fills ‘pfstat’ with it.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_FIND_CHMODE

Function:

Change file attributes.

Summary:

#include “cliapi.h”

void **rtsmb_cli_ez_chmode** (filename, attributes)

PFCHAR filename - the filename to change

int attributes - the new attributes of the file

Description:

This function changes the attributes of the named file. ‘Attributes’ can be one of the following: **RTSMB_O_RDONLY**, **RTSMB_O_WRONLY**, or **RTSMB_O_RDWR**.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_GET_FREE

Function:

Query disk size.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_get_free** (filename, ptotal_units, pfree_units, psectors_per_unit, pbytes_per_sector)

PFCHAR filename	- the filename whose disk you wish to query
PFINT ptotal_units	- return location for how many total units are on the disk
PFINT pfree_units	- return location for how many free units are on the disk
PFINT psectors_per_unit	- return location for how many sectors exist in each unit
PFINT pbytes_per_sector	- return location for how many bytes each sector takes up

Description:

This function queries the size of the disk on which the specified share resides. There must be at least a server and share specified in the filename. If there is more in the filename, it will be ignored.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_GET_CWD

Function:

Get the current working directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_get_cwd** (dest, size)

PFCHAR dest	- the return location for the current working directory
size_t size	- the size in characters of dest

Description:

This function fills ‘dest’ with the value of the current thread’s current working directory. If this filename is too long for ‘dest’, **RTSMB_CLI_EZ_NOT_ENOUGH_RESOURCES** will be returned.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTSMB_CLI_EZ_SET_CWD

Function:

Get the current working directory.

Summary:

#include “cliapi.h”

int **rtsmb_cli_ez_set_cwd** (dest)

PFCHAR dest	-	the new value for the current working directory

Description:

This function sets the current thread’s concept of the current working directory. If this filename is too large, **RTSMB_CLI_EZ_NOT_ENOUGH_RESOURCES** will be returned.

Returns:

Returns zero on success or a negative value on failure.

See Also:

See Appendix D for a list of error values.

RTIP 4.0

SMB Client Manual

Appendix A: Examples: EZ Layer Code

A P P E N D I X A

APPENDIX A - EZ LAYER CODE EXAMPLES

Note that for all of these examples, we are assuming that no network initialization must take place. For example, on Windows, each of these programs would need to include a WSAStartup() call.

ENUMERATING SERVERS

The following code example shows how to use the EZ layer to search for all servers in the network. It prints the name of each server to the screen.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_DSTAT dstat;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* Now, we can start searching. */
    r = rtsmb_cli_ez_find_first (“\\\\*”, &dstat);

    while (r == 1)
    {
    /* The returned filename may be Unicode if Unicode support is
    compiled in. */
        if (dstat.unicode)
        {
            printf (“Search data: name is %S (in Unicode)\n”,
            (unsigned short *) dstat.filename);
        }
        else
        {
            printf (“Search data: name is %s (in ASCII)\n”,
            dstat.filename);
        }

        /* Keep going until no more names. */
        r = rtsmb_cli_ez_find_next (&dstat);
    }

    /* We should always close the find. */
    rtsmb_cli_ez_find_close (&dstat);

    rtsmb_cli_shutdown ();
    return 0;
}
```

ENUMERATING SHARES

The following code example shows how to use the EZ layer to search for all shares on one server. It prints the name of each share to the screen.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_DSTAT dstat;
    char search_string [20];
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server name from the command line.  Let’s
    make sure the user gave one. */
    if (argc != 2)
    {
        printf (“Please provide exactly one server name as an
        argument.\n”);
        return 1;
    }
    else
    {
        sprintf (search_string, “\\\\%s\\*”, argv[1]);
    }

    /* Now, we can start searching. */
    r = rtsmb_cli_ez_find_first (search_string, &dstat);

    while (r == 1)
    {
    /* The returned filename may be Unicode if Unicode support is
    compiled in. */
        if (dstat.unicode)
        {
            printf (“Search data: name is %S (in Unicode)\n”,
            (unsigned short *) dstat.filename);
        }
        else
        {
            printf (“Search data: name is %s (in ASCII)\n”,
            dstat.filename);
        }

    /* Keep going until no more names. */
        r = rtsmb_cli_ez_find_next (&dstat);
    }

    /* We should always close the find. */
    rtsmb_cli_ez_find_close (&dstat);

    rtsmb_cli_shutdown ();
    return 0;
}
```

ENUMERATING FILES

The following code example shows how to use the EZ layer to search for all files in a share. It prints the name of each file to the screen.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_DSTAT dstat;
    char search_string [40];
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server and share names from the command
    line.  Let’s make sure the user gave them. */
    if (argc != 3)
    {
        printf (“Please provide one server name and one share
        name as arguments.\n”);
        return 1;
    }
    else
    {
        sprintf (search_string, “\\\\%s\\%s\\*”, argv[1], argv[2]);
    }

    /* Now, we can start searching. */
    r = rtsmb_cli_ez_find_first (search_string, &dstat);

    while (r == 1)
    {
    /* The returned filename may be Unicode if Unicode support is
    compiled in. */
        if (dstat.unicode)
        {
            printf (“Search data: name is %S (in Unicode)\n”,
            (unsigned short *) dstat.filename);
        }
        else
        {
            printf (“Search data: name is %s (in ASCII)\n”,
            dstat.filename);
        }

    /* Keep going until no more names. */
        r = rtsmb_cli_ez_find_next (&dstat);
    }

    /* We should always close the find. */
    rtsmb_cli_ez_find_close (&dstat);

    rtsmb_cli_shutdown ();
    return 0;
}
```

FILE I/O

The following code example shows how to use the EZ layer to create a file, write data to it, and then read it back to verify the contents. Warning — this code overwrites the contents of the file.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    char filename [100];
    char buffer [100];
    int fd;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server, share, path, and data string from
    the command line.  Let’s make sure the user gave them. */
    if (argc != 5)
    {
        printf (“Please provide a server, share, path, and data
        string as arguments.\n”);
        return 1;
    }
    else
    {
        sprintf (filename, “\\\\%s\\%s\\%.*s”, argv[1], argv[2], 50,
        argv[3]);
    }

    /* Now, we can open the file.  We want to create it if it does
    not exist, truncate it to 0 bytes if it does, open it with read and
    write permissions, and create it with read and write
    permissions if it does not exist.  */
    fd = rtsmb_cli_ez_open (filename, RTSMB_O_CREAT |
    RTSMB_O_RDWR | RTSMB_O_TRUNC, RTSMB_S_IWRITE |
    RTSMB_S_IREAD);
    if (fd < 0)
    {
        printf (“Could not open the file %s.\n”, filename);
        return 1;
    }

    /* Now, we want to write our command-line string to the file. */
    r = rtsmb_cli_ez_write (fd, argv[4], strlen (argv[4]));
    if (r < 0)
    {
        printf (“Could not write \”%s\” to file %s.\n”, argv[4],
        filename);
        return 1;
    }
    else
    {
        printf (“Wrote %i bytes to file %s.\n”, r, filename);
    }

    /* We are going to read the data back, so we have to set the
    file pointer at the beginning of the file. */
    r = rtsmb_cli_ez_seek (fd, 0, RTSMB_SEEK_SET);
    if (r < 0)
    {
        printf (“Could not seek to offset 0.\n”);
        return 1;
    }
    else
    {
        printf (“Sought to offset %i.\n”, r);
    }

    /* Now let’s read the data back. */
    r = rtsmb_cli_ez_read (fd, buffer, 100);
    if (r < 0)
    {
        printf (“Could not read data from file.\n”);
        return 1;
    }
    else
    {
        printf (“Read %i bytes.\n”, r);
    }

    /* Is it the same data? */
    if (memcmp (buffer, argv[4], strlen (argv[4]) < 100 ? strlen
    (argv[4]) : 100) == 0)
    {
        printf (“Yay!  Data written and data read are the same!\n”);
    }
    else
    {
        printf (“Uh, oh.  Data read is \”%s\”, while data written is
        \”%s\”.\n”, buffer, argv[4]);
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

RTIP 4.0

SMB Client Manual

Appendix B: Examples: Synchronous Session Layer Code

APPENDIX B

APPENDIX B - SYNCHRONOUS SESSION LAYER CODE EXAMPLES

Note that for all of these examples, we are assuming that no network initialization must take place. For example, on Windows, each of these programs would need to include a WSAStartup() call.

ENUMERATING SERVERS

The following code example shows how to use the synchronous session layer to search for all servers in the network. It prints the name of each server to the screen.

```
#include "cliapi.h"

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_SRVSTAT srvstat;
    char srvname [16];
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* Now, we can start searching.  We pass a block to fill out
    with internal data and two NULL's to indicate that it should
    use the default ip and broadcast ip. */
    r = rtsmb_cli_session_server_enum_start (&srvstat, NULL,
    NULL);
    if (r < 0)
    {
        printf ("Could not start the enumeration.\n");
        return 1;
    }

    /* Here's how getting the names works.  We cycle until data is
    ready or the search is over.  We then grab all the names we
    `can by calling enum_next_name() until we have exhausted
    its names.  We then go back to cycling again.  We repeat this
    until enum_cycle() returns CSSN_RV_END_OF_SEARCH. */
    do
    {
        /* We need to allow the session to do its thing.  Here we
        cycle with no timeout until an error occurs, the search is
        over, or data is ready. */
        do
        {
            r = rtsmb_cli_session_server_enum_cycle (&srvstat, -1);
        }
        while (r == 0);

        /* Is the search over gracefully? */
        if (r == CSSN_RV_END_OF_SEARCH)
        {
            break;
        }
        /* Some error. */
        else if (r != CSSN_RV_SEARCH_DATA_READY)
        {
            printf ("Error when cycling.\n");
            return 1;
        }

        /* OK.  So data is ready to be read. */
        do
        {
            /* Tell the enumeration to put the next name in the 16
            char array srvname.  Server names are always in
            ASCII. */
            r = rtsmb_cli_session_server_enum_next_name
            (&srvstat, srvname);

            if (r == CSSN_RV_SEARCH_DATA_READY)
            {
                printf ("Search data: name is %s (in ASCII)\n",
                srvname);
            }
        }
        while (r == CSSN_RV_SEARCH_DATA_READY);

        if (r != CSSN_RV_END_OF_SEARCH)
        {
            /* We must have had an error while getting names. */
            printf ("Error while getting names.\n");
            return 1;
        }

        /* OK.  That's one iteration down.  Now, we go back to
        cycle again and start the process over. */
    }
    while (1);

    /* We should always close the enum. */
    rtsmb_cli_session_server_enum_close (&srvstat);

    rtsmb_cli_shutdown ();
    return 0;
}
```

ENUMERATING SHARES

The following code example shows how to use the synchronous session layer to search for all shares on one server. It prints the name of each share to the screen.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_SSTAT sstat;
    int sid;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server name from the command line.  Let’s
    make sure the user gave one. */
    if (argc != 2)
    {
        printf (“Please provide exactly one server name as an
        argument.\n”);
        return 1;
    }

    /* Now, we create a session.  argv[1] is the server name,
    TRUE means we want blocking mode, and NULL means we
    want to use the default broadcast ip. */
    r = rtsmb_cli_session_new_with_name (argv[1], TRUE, NULL,
    &sid);
    if (r < 0)
    {
        printf (“Error creating session with server %s.\n”,
        argv[1]);
        return 1;
    }

    /* Start off the share find.  We do not need to log in or connect
    to shares. */
    r = rtsmb_cli_session_share_find_first (sid, &sstat);

    while (r == CSSN_RV_SEARCH_DATA_READY)
    {
        /* All share names are in ASCII. */
        printf (“Search data: name is %s (in ASCII)\n”, sstat.name);

        /* Grab the next one. */
        r = rtsmb_cli_session_share_find_next (sid, &sstat);
    }

    /* We should always close the find. */
    rtsmb_cli_session_share_find_close (sid, &sstat);

    if (r != CSSN_RV_END_OF_SEARCH)
    {
        /* Some error occurred. */
        printf (“Error while finding shares.\n”);
        return 1;
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

ENUMERATING FILES

The following code example shows how to use the synchronous session layer to search for all files in a share. It prints the name of each file to the screen.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_DSTAT dstat;
    int sid;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server and share names from the command
    line.  Let’s make sure the user gave them. */
    if (argc != 3)
    {
        printf (“Please provide one server name and one share
        name as arguments.\n”);
        return 1;
    }

    /* Now, we create a session.  argv[1] is the server name,
    TRUE means we want blocking mode, and NULL means we
    want to use the default broadcast ip. */
    r = rtsmb_cli_session_new_with_name (argv[1], TRUE, NULL,
    &sid);
    if (r < 0)
    {
        printf (“Error creating session with server %s.\n”,
        argv[1]);
        return 1;
    }

    /* Now, we log in a user.  Since we don’t have any particular
    credentials, we will just use the name “anonymous” and a
    null password. */
    r = rtsmb_cli_session_logon_user (sid, “anonymous”, “”);
    if (r < 0)
    {
        printf (“Error logging on user anonymous.\n”);
        return 1;
    }

    /* Now, we connect to the share we want.  Since we don’t
    have any particular credentials, we will just use a null
    password. */
    r = rtsmb_cli_session_connect_share (sid, argv[2], “”);
    if (r < 0)
    {
        printf (“Error connecting to share %s.\n”, argv[2]);
        return 1;
    }

    /* Start off the file find. */
    r = rtsmb_cli_session_find_first (sid, argv[2], “\\*”, &dstat);

    while (r == CSSN_RV_SEARCH_DATA_READY)
    {
        /* File names may be in ASCII or Unicode. */
        if (dstat.unicode)
        {
            printf ("Search data: name is %S (in Unicode)\n",
            dstat.filename);
        }
        else
        {
            printf ("Search data: name is %s (in ASCII)\n",
            dstat.filename);
        }

        /* Grab the next one. */
        r = rtsmb_cli_session_find_next (sid, &dstat);
    }

    /* We should always close the find. */
    rtsmb_cli_session_find_close (sid, &dstat);

    if (r != CSSN_RV_END_OF_SEARCH)
    {
        /* Some error occurred. */
        printf ("Error while finding files.\n");
        return 1;
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

FILE I/O

The following code example shows how to use the synchronous session layer to create a file, write data to it, and then read it back to verify the contents. Warning — this code overwrites the contents of the file.

```
#include "cliapi.h"

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    char buffer [100];
    unsigned short written, read;
    unsigned int offset;
    int fd;
    int sid;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server, share, path, and data string from
    the command line.  Let's make sure the user gave them. */
    if (argc != 5)
    {
        printf ("Please provide a server, share, path, and data
        string as arguments.\n");
        return 1;
    }

    /* Now, we create a session.  argv[1] is the server name,
    TRUE means we want blocking mode, and NULL means we
    want to use the default broadcast ip. */
    r = rtsmb_cli_session_new_with_name (argv[1], TRUE, NULL,
    &sid);
    if (r < 0)
    {
        printf ("Error creating session with server %s.\n",
        argv[1]);
        return 1;
    }

    /* Now, we log in a user.  Since we don't have any particular
    credentials, we will just use the name "anonymous" and a
    null password. */
    r = rtsmb_cli_session_logon_user (sid, "anonymous", "");
    if (r < 0)
    {
        printf ("Error logging on user anonymous.\n");
        return 1;
    }

    /* Now, we connect to the share we want.  Since we don't
    have any particular credentials, we will just use a null
    password. */
    r = rtsmb_cli_session_connect_share (sid, argv[2], "");
    if (r < 0)
    {
        printf ("Error connecting to share %s.\n", argv[2]);
        return 1;
    }

    /* Now, we can open the file.  We want to create it if it does
    not exist, truncate it to 0 bytes if it does, open it with read and
    write permissions, and create it with read and write
    permissions if it does not exist.  We receive the file descriptor
    by passing a pointer to 'fd'. */
    r = rtsmb_cli_session_open (sid, argv[2], argv[3],
    RTSMB_O_CREAT | RTSMB_O_RDWR | RTSMB_O_TRUNC,
    RTSMB_S_IWRITE | RTSMB_S_IREAD, &fd);
    if (r < 0)
    {
        printf ("Could not open the file %s on share %s.\n", argv[3],
        argv[2]);
        return 1;
    }

    /* Now, we want to write our command-line string to the file.
    We receive the number of bytes written by passing a pointer
    to 'written'. */
    r = rtsmb_cli_session_write (sid, fd, argv[4], (unsigned short)
    strlen (argv[4]), &written);
    if (r < 0)
    {
        printf ("Could not write \"%s\" to file %s.\n", argv[4],
        argv[3]);
    }
    else
    {
        printf ("Wrote %i bytes to file %s.\n", written, argv[3]);
    }

    /* We are going to read the data back, so we have to set the
    file pointer at the beginning of the file.  We receive the new
    offset by passing a pointer to 'offset'. */
    r = rtsmb_cli_session_seek (sid, fd, 0, RTSMB_SEEK_SET,
    &offset);
    if (r < 0)
    {
        printf ("Could not seek to offset 0.\n");
    }
    else
    {
        printf ("Sought to offset %i.\n", offset);
    }

    /* Now let's read the data back.  We receive the number of
    bytes read by passing a pointer to 'read'. */
    r = rtsmb_cli_session_read (sid, fd, buffer, 100, &read);
    if (r < 0)
    {
        printf ("Could not read data from file.\n");
    }
    else
    {
        printf ("Read %i bytes.\n", read);
    }

    /* Is it the same data? */
    if (memcmp (buffer, argv[4], strlen (argv[4]) < 100 ? strlen
    (argv[4]) : 100) == 0)
    {
        printf ("Yay!  Data written and data read are the same!\n");
    }
    else
    {
        printf ("Uh, oh.  Data read is \"%s\", while data written is
        \"%s\".\n", buffer, argv[4]);
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

RTIP 4.0

SMB Client Manual

Appendix C: Examples: Asynchronous Session Layer Code

A P P E N D I X C

APPENDIX C - ASYNCHRONOUS SESSION LAYER CODE EXAMPLES

Note that for all of these examples, we are assuming that no network initialization must take place. For example, on Windows, each of these programs would need to include a WSAStartup() call.

ENUMERATING SERVERS

The following code example shows how to use the asynchronous session layer to search for all servers in the network. It prints the name of each server to the screen. This is the asynchronous code example most similar to its synchronous version.

```
#include “cliapi.h”

int main (int argc, char *argv[])
{
    /* Some predefined information about our network. Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_SRVSTAT srvstat;
    char srvname [16];
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* Now, we can start searching. We pass a block to fill out
    with internal data, we pass two NULL’s to indicate that it
    should use the default ip and broadcast ip. */
    r = rtsmb_cli_session_server_enum_start (&srvstat, NULL,
    NULL);
    if (r < 0)
    {
        printf (“Could not start the enumeration.\n”);
        return 1;
    }

    /* Here’s how getting the names works. We cycle until data is
    ready or the search is over. We then grab all the names we
    can by calling enum_next_name() until we have exhausted
    its names. We then go back to cycling again. We repeat this
    until enum_cycle() returns CSSN_RV_END_OF_SEARCH. */
    do
    {
    /* We need to allow the session to do its thing. Here we cycle
    with a 10-millisecond timeout until an error occurs, the search
    is over, or data is ready. */
        do
        {
            r = rtsmb_cli_session_server_enum_cycle (&srvstat, 10);

            if (r == 0)
            {
                printf (“In the middle of cycling. I could be doing
                something else while I wait now.\n”);
            }
        }
        while (r == 0);

        /* Is the search over gracefully? */
        if (r == CSSN_RV_END_OF_SEARCH)
        {
            break;
        }
        /* Some error. */
        else if (r != CSSN_RV_SEARCH_DATA_READY)
        {
            printf (“Error when cycling.\n”);
            return 1;
        }

        /* OK. So data is ready to be read. */
        do
        {
            /* Tell the enumeration to put the next name in the 16
            char array srvname. Server names are always in
            ASCII. */
            r = rtsmb_cli_session_server_enum_next_name
            (&srvstat, srvname);

            if (r == CSSN_RV_SEARCH_DATA_READY)
            {
                printf (“Search data: name is %s (in ASCII)\n”,
                srvname);
            }
        }
        while (r == CSSN_RV_SEARCH_DATA_READY);

        if (r != CSSN_RV_END_OF_SEARCH)
        {
            /* We must have had an error while getting names. */
            printf (“Error while getting names.\n”);
            return 1;
        }

        /* OK. That’s one iteration down. Now, we go back to
        cycle again and start the process over. */
    }
    while (1);

    /* We should always close the enum. */
    rtsmb_cli_session_server_enum_close (&srvstat);

    rtsmb_cli_shutdown ();
    return 0;
}
```

ENUMERATING SHARES

The following code example shows how to use the asynchronous session layer to search for all shares on one server. It prints the name of each share to the screen.

```
#include “cliapi.h”

/* This is a simple callback function for wait_on_job() to use.
When we get the response, we set data to be the return value. */
void mark_rv (int job, int rv, void *data)
{
    int *idata = (int *) data;

    *idata = rv;
}

/* This function will wait until the job ‘job’ is complete, never
blocking for more than 10 milliseconds. */
int wait_on_job (int sid, int job)
{
    int rv = CSSN_RV_INVALID_RV;
    int r;
    rtsmb_cli_session_set_job_callback (sid, job, mark_rv, &rv);

    while (rv == CSSN_RV_INVALID_RV)
    {
        r = rtsmb_cli_session_cycle (sid, 10);
        if (r < 0)
        {
            printf (“Something is wrong with the session.
            Bailing.\n”);
            return r;
        }

        if (rv == CSSN_RV_INVALID_RV)
        {
            printf (“In the middle of cycling.  I could be doing
            something else while I wait now.\n”);
        }
    }

    return rv;
}

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_SSTAT sstat;
    int sid;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server name from the command line.  Let’s
    make sure the user gave one. */
    if (argc != 2)
    {
        printf (“Please provide exactly one server name as an
    argument.\n”);
        return 1;
    }

    /* Now, we create a session.  argv[1] is the server name,
    FALSE means we want non-blocking mode, and NULL means
    we want to use the default broadcast ip. */
    r = rtsmb_cli_session_new_with_name (argv[1], FALSE,
    NULL, &sid);
    if (r < 0)
    {
        printf (“Error creating session with server %s.\n”,
        argv[1]);
        return 1;
    }

    /* OK.  Now, we should wait until the session is settled. */
    r = wait_on_job (sid, r);
    if (r < 0)
    {
        printf (“Error upon create session response.\n”);
        return 1;
    }

    /* Start off the share find.  We do not need to log in or connect
    to shares. */
    r = rtsmb_cli_session_share_find_first (sid, &sstat);
    if (r < 0)
    {
        printf (“Error when trying to start share find.\n”);
        return 1;
    }

    r = wait_on_job (sid, r);

    while (r == CSSN_RV_SEARCH_DATA_READY)
    {
        /* All share names are in ASCII. */
        printf (“Search data: name is %s (in ASCII)\n”, sstat.name);

        /* Grab the next one. */
        r = rtsmb_cli_session_share_find_next (sid, &sstat);

        /* share_find_first does not need to cycle. */
    }

    /* We should always close the find. */
    rtsmb_cli_session_share_find_close (sid, &sstat);

    if (r != CSSN_RV_END_OF_SEARCH)
    {
        /* Some error occurred. */
        printf (“Error while finding shares.\n”);
        return 1;
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

ENUMERATING FILES

The following code example shows how to use the asynchronous session layer to search for all files in a share. It prints the name of each file to the screen.

```
#include “cliapi.h”

/* This is a simple callback function for wait_on_job() to use.
When we get the response, we set data to be the return value. */
void mark_rv (int job, int rv, void *data)
{
    int *idata = (int *) data;

    *idata = rv;
}

/* This function will wait until the job ‘job’ is complete, never
blocking for more than 10 milliseconds. */
int wait_on_job (int sid, int job)
{
    int rv = CSSN_RV_INVALID_RV;
    int r;
    rtsmb_cli_session_set_job_callback (sid, job, mark_rv, &rv);

    while (rv == CSSN_RV_INVALID_RV)
    {
        r = rtsmb_cli_session_cycle (sid, 10);
        if (r < 0)
        {
            printf (“Something is wrong with the session.
            Bailing.\n”);
            return r;
        }

        if (rv == CSSN_RV_INVALID_RV)
        {
            printf (“In the middle of cycling. I could be doing
            something else while I wait now.\n”);
        }
    }

    return rv;
}

int main (int argc, char *argv[])
{
    /* Some predefined information about our network. Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    RTSMB_CLI_SESSION_DSTAT dstat;
    int sid;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server and share names from the command
    line. Let’s make sure the user gave them. */
    if (argc != 3)
    {
        printf (“Please provide one server name and one share
        name as arguments.\n”);
        return 1;
    }

/* Now, we create a session. argv[1] is the server name,
FALSE means we want non-blocking mode, and NULL means
we want to use the default broadcast ip. */
r = rtsmb_cli_session_new_with_name (argv[1], FALSE,
NULL, &sid);
if (r < 0)
{
    printf (“Error creating session with server %s.\n”, argv[1]);
    return 1;
}

/* OK. Now, we should wait until the session is settled. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon create session response.\n”);
    return 1;
}

/* Now, we log in a user. Since we don’t have any particular
credentials, we will just use the name “anonymous” and a
null password. */
r = rtsmb_cli_session_logon_user (sid, “anonymous”, “”);
if (r < 0)
{
    printf (“Error logging on user anonymous.\n”);
    return 1;
}

/* OK. Now, we should wait until the user is logged in. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon logon response.\n”);
    return 1;
}

/* Now, we connect to the share we want. Since we don’t
have any particular credentials, we will just use a null
password. */
r = rtsmb_cli_session_connect_share (sid, argv[2], “”);
if (r < 0)
{
    printf (“Error connecting to share %s.\n”, argv[2]);
    return 1;
}

/* OK. Now, we should wait until the share is connected. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon share connect response.\n”);
    return 1;
}

/* Start off the file find. */
r = rtsmb_cli_session_find_first (sid, argv[2], “\\*”, &dstat);
if (r < 0)
{
    printf (“Error starting find job.\n”);
    return 1;
}

/* OK. Now, we should wait until the share is connected. */
r = wait_on_job (sid, r);

while (r == CSSN_RV_SEARCH_DATA_READY)
{
    /* File names may be in ASCII or Unicode. */
    if (dstat.unicode)
    {
            printf (“Search data: name is %S (in Unicode)\n”,
            dstat.filename);
        }
        else
        {
            printf (“Search data: name is %s (in ASCII)\n”,
            dstat.filename);
        }

        /* Grab the next one. */
        r = rtsmb_cli_session_find_next (sid, &dstat);

        /* find_next may or may not need to cycle.  Only cycle if
        we have a job number. */
        if (r >= 0)
        {
            r = wait_on_job (sid, r);
        }
    }

    /* We should always close the find. */
    rtsmb_cli_session_find_close (sid, &dstat);

    if (r != CSSN_RV_END_OF_SEARCH)
    {
        /* Some error occurred. */
        printf (“Error while finding files.\n”);
        return 1;
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

FILE I/O

The following code example shows how to use the asynchronous session layer to create a file, write data to it, and then read it back to verify the contents. Warning — this code overwrites the contents of the file.

```
#include “cliapi.h”

/* This is a simple callback function for wait_on_job() to use.
When we get the response, we set data to be the return value. */
void mark_rv (int job, int rv, void *data)
{
    int *idata = (int *) data;

    *idata = rv;
}

/* This function will wait until the job ‘job’ is complete, never
blocking for more than 10 milliseconds. */
int wait_on_job (int sid, int job)
{
    int rv = CSSN_RV_INVALID_RV;
    int r;
    rtsmb_cli_session_set_job_callback (sid, job, mark_rv, &rv);

    while (rv == CSSN_RV_INVALID_RV)
    {
        r = rtsmb_cli_session_cycle (sid, 10);
        if (r < 0)
        {
            printf (“Something is wrong with the session.
            Bailing.\n”);
            return r;
        }

        if (rv == CSSN_RV_INVALID_RV)
        {
            printf (“In the middle of cycling.  I could be doing
            something else while I wait now.\n”);
        }
    }

    return rv;
}

int main (int argc, char *argv[])
{
    /* Some predefined information about our network.  Ideally
    would be gotten dynamically. */
    unsigned char my_ip [] = {192, 168, 1, 100};
    unsigned char my_mask [] = {255, 255, 255, 0};

    /* Some data that we will use later. */
    char buffer [100];
    unsigned short written, read;
    unsigned int offset;
    int fd;
    int sid;
    int r;

    /* Here we set the default ip and subnet mask for RTSMB.
    This needs to be done before any RTSMB Client calls. */
    rtsmb_cli_init (my_ip, my_mask);

    /* We will grab the server, share, path, and data string from
    the command line.  Let’s make sure the user gave them. */
    if (argc != 5)
    {
        printf (“Please provide a server, share, path, and data
        string as arguments.\n”);
        return 1;
}

/* Now, we create a session.  argv[1] is the server name,
FALSE means we want non-blocking mode, and NULL means
we want to use the default broadcast ip. */
r = rtsmb_cli_session_new_with_name (argv[1], FALSE,
NULL, &sid);
if (r < 0)
{
    printf (“Error creating session with server %s.\n”,
argv[1]);
    return 1;
}

/* OK.  Now, we should wait until the session is settled. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon create session response.\n”);
    return 1;
}

/* Now, we log in a user.  Since we don’t have any particular
credentials, we will just use the name “anonymous” and a
null password. */
r = rtsmb_cli_session_logon_user (sid, “anonymous”, “”);
if (r < 0)
{
    printf (“Error logging on user anonymous.\n”);
    return 1;
}

/* OK.  Now, we should wait until the user is logged in. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon logon response.\n”);
    return 1;
}

/* Now, we connect to the share we want.  Since we don’t have
any particular credentials, we will just use a null password. */
r = rtsmb_cli_session_connect_share (sid, argv[2], “”);
if (r < 0)
{
    printf (“Error connecting to share %s.\n”, argv[2]);
    return 1;
}

/* OK.  Now, we should wait until the share is connected. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon share connect response.\n”);
    return 1;
}

/* Now, we can open the file.  We want to create it if it does
not exist, truncate it to 0 bytes if it does, open it with read and
write permissions, and create it with read and write
permissions if it does not exist.  We receive the file descriptor
by passing a pointer to ‘fd’. */
r = rtsmb_cli_session_open (sid, argv[2], argv[3],
RTSMB_O_CREAT | RTSMB_O_RDWR | RTSMB_O_TRUNC,
RTSMB_S_IWRITE | RTSMB_S_IREAD, &fd);
if (r < 0)
{
    printf (“Could not open the file %s on share %s.\n”, argv[3],
    argv[2]);
    return 1;
}

/* OK.  Now, we should wait until the file is opened. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon open response.\n”);
    return 1;
}

/* Now, we want to write our command-line string to the file.
We receive the number of bytes written by passing a pointer
to ‘written’. */
r = rtsmb_cli_session_write (sid, fd, argv[4], (unsigned short)
strlen (argv[4]), &written);
if (r < 0)
{
    printf (“Could not write \”%s\” to file %s.\n”, argv[4],
    argv[3]);
}

/* OK.  Now, we should wait until the data is written. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon write response.\n”);
    return 1;
}

/* Now ‘written’ contains the number of bytes. */
printf (“Wrote %i bytes to file %s.\n”, written, argv[3]);

/* We are going to read the data back, so we have to set the
file pointer at the beginning of the file.  We receive the new
offset by passing a pointer to ‘offset’. */
r = rtsmb_cli_session_seek (sid, fd, 0, RTSMB_SEEK_SET,
&offset);
if (r < 0)
{
    printf (“Could not seek to offset 0.\n”);
}

/* OK.  Now, we should wait until seek is done. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon seek response.\n”);
    return 1;
}

/* Now ‘offset’ contains the new offset. */
printf (“Sought to offset %i.\n”, offset);

/* Now let’s read the data back.  We receive the number of
bytes read by passing a pointer to ‘read’. */
r = rtsmb_cli_session_read (sid, fd, buffer, 100, &read);
if (r < 0)
{
    printf (“Could not read data from file.\n”);
}

/* OK.  Now, we should wait until seek is done. */
r = wait_on_job (sid, r);
if (r < 0)
{
    printf (“Error upon read response.\n”);
    return 1;
}

/* Now ‘read’ contains the number of bytes read. */
printf (“Read %i bytes.\n”, read);
```

```
    /* Is it the same data? */
    if (memcmp (buffer, argv[4], strlen (argv[4]) < 100 ? strlen
    (argv[4]) : 100) == 0)
    {
        printf ("Yay!  Data written and data read are the same!\n");
    }
    else
    {
        printf ("Uh, oh.  Data read is \"%s\", while data written is
        \"%s\".\n", buffer, argv[4]);
    }

    rtsmb_cli_shutdown ();
    return 0;
}
```

RTIP 4.0

SMB Client Manual

Appendix D: Error Values

APPENDIX D

THE EZ API LAYER ERROR VALUES

/* This is returned if the filename passed in could not be parsed. */

#define RTSMB_CLI_EZ_INVALID_PATH -2

/* This is returned if we have problems connecting to a server. */

#define RTSMB_CLI_EZ_COULD_NOT_CONNECT -3

/* This is returned if there was some generic error on the server side. */

#define RTSMB_CLI_EZ_SESSION_ERROR -5

/* This is returned if we did not recognize the file descriptor. */

#define RTSMB_CLI_EZ_BAD_FD -6

/* This is returned if there was not a server specified in the filename, and the function requires one. */

#define RTSMB_CLI_EZ_NO_SERVER_SPECIFIED -7

/* This is returned if there was not a share specified in the filename, and the function requires one. */

#define RTSMB_CLI_EZ_NO_SHARE_SPECIFIED -8

/* This is returned if there was not a filename path specified in the filename, and the function requires one. */

#define RTSMB_CLI_EZ_NO_FILENAME_SPECIFIED -9

/* This is returned if a buffer is not big enough or too many searches are trying to be run at once. */

#define RTSMB_CLI_EZ_NOT_ENOUGH_RESOURCES -10

/* This is returned if you are trying to rename across sessions/shares. */

#define RTSMB_CLI_EZ_NOT_SAME_SESSION -11

/* This is returned if the file you were trying to use did not exist on the server. */

#define RTSMB_CLI_EZ_FILE_NOT_FOUND -12

/* This is returned if you do not have the permissions to attempt some file operation (like write to a read only file). */

#define RTSMB_CLI_EZ_BAD_PERMISSIONS -13

SESSION API LAYER ERROR VALUES

/* This is returned if everything is good */

#define RTSMB_CLI_SSN_RV_OK 0

/* This is returned if something was malformed on the wire */

#define RTSMB_CLI_SSN_RV_MALFORMED -1

/*This is returned if you need to try again later */

#define RTSMB_CLI_SSN_RV_LATER -2

This is returned if session is untenable and should be closed */

#define RTSMB_CLI_SSN_RV_DEAD -3 /*

/* This is returned if job id is invalid */

#define RTSMB_CLI_SSN_RV_BAD_JOB -5

This is returned if invalid netbios name passed */

#define RTSMB_CLI_SSN_RV_BAD_NAME -19 /*

This is returned if argument to function is out of range */

#define RTSMB_CLI_SSN_RV_BAD_ARGS -6 /*

/* This is returned if too many jobs waiting */

#define RTSMB_CLI_SSN_RV_TOO_MANY_JOBS -7

/* This is returned if too many users logged on */

#define RTSMB_CLI_SSN_RV_TOO_MANY_USERS -8

/* This is returned if too many shares already connected */

#define RTSMB_CLI_SSN_RV_TOO_MANY_SHARES -11

/* This is returned if too many fids already open */

#define RTSMB_CLI_SSN_RV_TOO_MANY_FIDS -13

/* This is returned if too many searches already open */

#define RTSMB_CLI_SSN_RV_TOO_MANY_SEARCHES -15

/* This is returned if bad share name */

#define RTSMB_CLI_SSN_RV_BAD_SHARE -12

/* This is returned if bad file name */

#define RTSMB_CLI_SSN_RV_BAD_FILENAME -21

/* This is returned if bad fid */

#define RTSMB_CLI_SSN_RV_BAD_FID -14

/* This is returned if bad session id */

#define RTSMB_CLI_SSN_RV_BAD_SID -23

/* This is returned if bad search struct passed in */

#define RTSMB_CLI_SSN_RV_BAD_SEARCH -16

/* This is returned if data is available from a search struct */

#define RTSMB_CLI_SSN_RV_SEARCH_DATA_READY -17

/* This is returned if no more data from this search */

#define RTSMB_CLI_SSN_RV_END_OF_SEARCH -18

/* This is returned if already connected to a share */

#define RTSMB_CLI_SSN_RV_ALREADY_CONNECTED -20

/* This is returned if not enough search structs to hand out */

```
#define RTSMB_CLI_SSN_RV_NOT_ENOUGH_RESOURCES -
22
```

/* This is returned if no user logged on */

```
#define RTSMB_CLI_SSN_RV_NO_USER -10
```

/* This is returned if an error occurred on server for a particular smb */

```
#define RTSMB_CLI_SSN_RV_SMB_ERROR -50
```

/* This is returned if an error occurred on server for a particular smb */

```
#define RTSMB_CLI_SSN_RV_FILE_NOT_FOUND -51
```

/* This is returned if an error occurred on server for a particular smb */

```
#define RTSMB_CLI_SSN_RV_BAD_PERMISSIONS -52
```

/* This is returned if this is guaranteed to never be used as an rv value */

```
#define RTSMB_CLI_SSN_RV_INVALID_RV -100
```