ThinShare

SMB Client Suite Manual

v1.50

15 May 2008

Embeo, Inc.

ThinShare: SMB Client Suite Manual

Embeo, Inc.

Published 05/15/2008

Table of Contents

Part I. Getting Started

Table of Contents

Chapter 1. Introduction to SMB

The SMB protocol, or *Windows File Sharing* as it is more commonly known, is the de facto standard for file and printer sharing in use today. An SMB server makes file and printer resources available over a computer network. Once set up, SMB client software can access these resources and use them according to their purpose. The most recent version has been a built-in component to all versions of Microsoft Windows since the release of Windows 95. There have also been many software suites written, the most popular being Samba [http://www.samba.org], to expand the compatibility to any UNIX-like operating system (including Linux and Mac OS X). As a result, SMB is ubiquitous-- operating throughout corporate and home networks.

ThinShare is an SMB client implementation. ThinShare supports communications with Windows 9x/NT4 and later, either in workgroup or domain configurations. Likewise, it is compatible with all versions of Samba. It must be compiled with Rabbit Semiconductor's Dynamic C version 7.05 or later development system to any TCP/IP enabled Rabbit 2000/3000 system. Note that Dynamic C version 8.30 or later is required for using DHCP to acquire a WINS server IP address. If this is not needed, standard DHCP is still supported with the previous Dynamic C versions.

ThinShare offers the following services to embedded applications:

- Standard File Operations: create, open, close, read, write, seek, get/set attributes, rename
- Standard Directory Operations: create, delete, wildcard search with attribute filter
- Standard Print Services: open/write/close print spool files
- User Authentication: plain-text, encrypted LM session key, encrypted NT session key
- Server Operations: get browse server, list servers in workgroup/domain, list shares on server

ThinShare also offers SMB Browse service functionality that enables machine icons to appear in Network Neighborhood/My Network Places of Windows Explorer.

Embeo's ThinShare SMB Client Suite uses the NetBIOS transport to communicate with other machines on the network. The NetBIOS layer offers IPC services and host name resolution to applications such as SMB. The included NetBIOS layer includes support for all features required for proper SMB functionality. It is an RFC1001/1002 compliant client implementation of the NetBIOS standard. It offers many configuration options which are documented in Section 11.1, “NetBIOS Options”.

Chapter 2. Installing ThinShare

1. Insert the installation media and run the install file `ersmbinst.exe`.
2. Select components to install. Options are:
   - Documentation and Samples
   - Start Menu Shortcuts
   - Add entries in LIB.DIR
   - Integrate Install with Dynamic C
3. Select destination directory.
4. (If 'Add entries in LIB.DIR' is selected in Step 2. If not, LIB.DIR must be manually edited) The installer is capable of detecting installed versions of Dynamic C, and it can update your LIB.DIR for you. Just select the "Add entry in LIB.DIR" option, and click YES when prompted to use the version of Dynamic C listed. If you have more than one version of Dynamic C installed and do not see the version listed, click NO and the installer will ask if you would like to use the next version detected. When the installer runs out of options, clicking NO results in being prompted for a LIB.DIR to use.
5. Done! Open up a sample program and try connecting to a machine!

   Sample programs are located in the \Samples\ThinShare folder relative to the Dynamic C installation (or ThinShare\Samples if integrate with Dynamic C is not selected).

Adding Entries to LIB.DIR

The ThinShare SMB Client will not work unless the .lib files are added to your LIB.DIR file used by Dynamic C. Refer to the Dynamic C user manual and LIB.DIR (in your Dynamic C installation directory) for more details.

If you would like to manually modify your LIB.DIR(s), simply deselect the option during installation and remember where you installed the library and add the following lines to the end of your LIB.DIR (fill in with the correct path):

Integrate install with Dynamic C selected:

C:\Program Files\DCRABBIT_10.21\Rabbit4000\Lib\ThinShare\des.lib
C:\Program Files\DCRABBIT_10.21\Rabbit4000\Lib\ThinShare\md4.lib
C:\Program Files\DCRABBIT_10.21\Rabbit4000\Lib\ThinShare\netbios.lib
C:\Program Files\DCRABBIT_10.21\Rabbit4000\Lib\ThinShare\smb_config.lib
C:\Program Files\DCRABBIT_10.21\Rabbit4000\Lib\ThinShare\smb.lib

Integrate install with Dynamic C not selected:

C:\Program Files\Embeo\ThinShare\des.lib
C:\Program Files\Embeo\ThinShare\md4.lib
C:\Program Files\Embeo\ThinShare\netbios.lib
C:\Program Files\Embeo\ThinShare\smb_config.lib
C:\Program Files\Embeo\ThinShare\smb.lib

Chapter 3. Sample Programs

This section describes the configurations for both Windows and Linux that are needed to run the sample programs. The sample programs are described next.

3.1. Introduction

The samples all require the use of an SMB share with read-write access. ThinShare uses encrypted passwords (LM or NTLM) to connect to the remote machine.

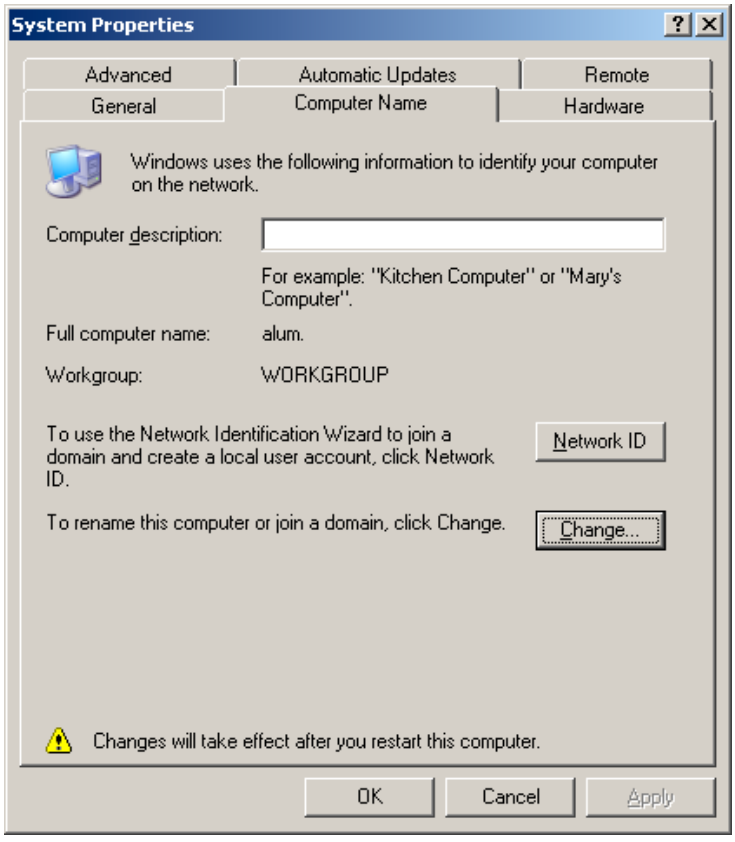
Note: Some servers (Windows 2003 Server and above) may be set by default to reject this form of encrypted passwords and use high encryption only. (See Section 3.2.2, “Windows 2003 Server Settings” [7])

3.2. Windows/Linux PC Server Configuration

In order to use the samples you must have a server on the same network as the rabbit. The server must have a shared folder with an associated username and password that has access.

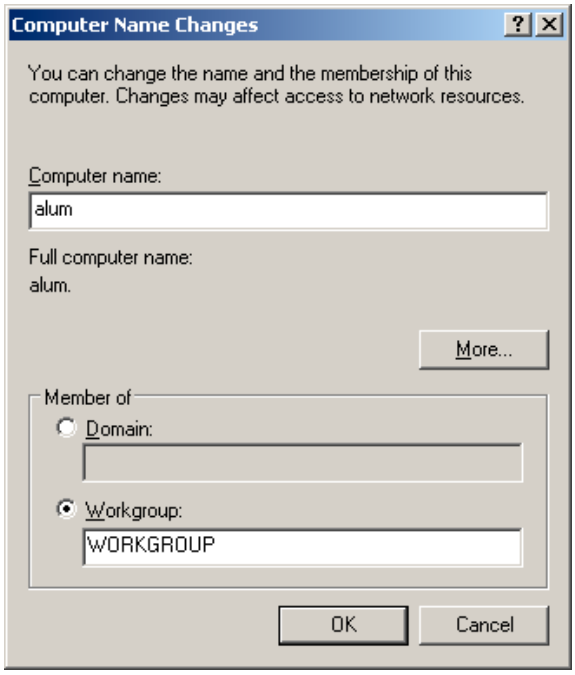
3.2.1. Windows Server Setup

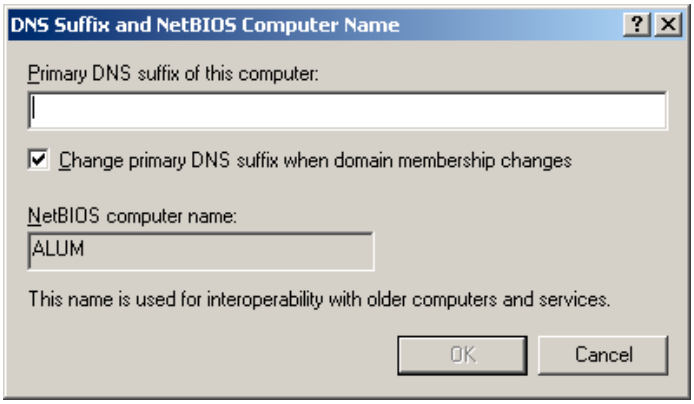
First the server's name must be determined. The name can be acquired from System Properties (right click My Computer -> Properties -> Network Identification) or by typing ipconfig /all in a command window. The name of the server in this case is 'alum'.



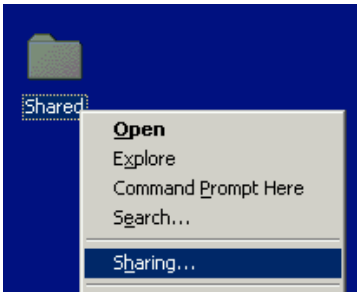
```
C:\Documents and Settings\syplex>ipconfig /all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : alum
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Broadcast
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
```

Note: The Full computer name may not be the same as the NetBIOS name. To get the NetBIOS name click the Change or Properties button in the Network Identification tab, and then click the More button. The name is listed under NetBIOS computer name.

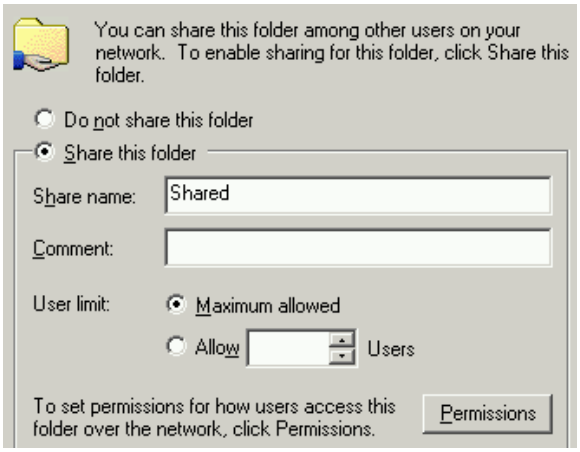




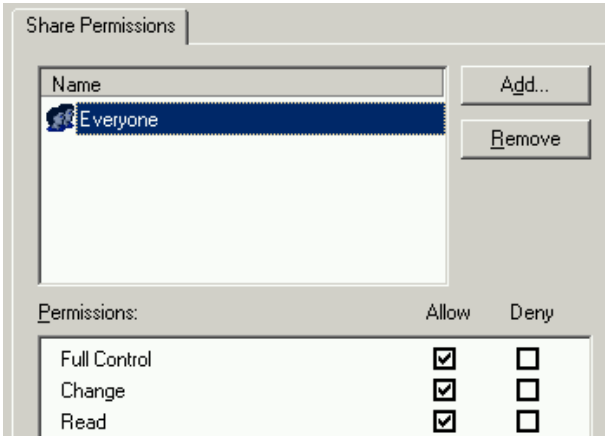
In explorer, navigate to the folder you wish to share out, right click and select *Sharing*.



Make a note of the Share name; this name along with the server's name is used to make an SMB connection. The share name in this case is 'Shared'.



Make sure that the the username used to access the shared folder has read/write access to the share. It is a good idea to create a new user account on the server or domain controller in order to limit access to certain shares or files.



3.2.2. Windows 2003 Server Settings

Default security settings in Windows 2003 require high encryption for SMB authentication. So by default, ThinShare clients will not be able to connect to shares on a Windows 2003 domain. In order to enable ThinShare clients to connect the following settings must be made:

Domain Controller Security Settings -> Local Policies -> Security Options

Digitally Sign communications (always) *DISABLED*

Digitally Sign communications (if client agrees) *ENABLED*

Network Security

LAN Manager authentication level **MUST NOT** be set to the following (i.e., LM or NTLM must be allowed):

[X] Send NTLMv2 resonse only\refuse LM

[X] Send NTLMv2 resonse only\refuse LM & NTLM

These settings may take some time to take effect; run the command **gpupdate** to apply the settings immediately.

Note: If you disable NTLM make sure you disable it in ThinShare by defining the proper macro:

```
// To disable NTLM keys
#define _SMB_AUTH_USE_NT 0
// To disable LM keys
#define _SMB_AUTH_USE_LM 0
// To use both NTLM and LM keys (not necessary, defaults to 1)
#define _SMB_AUTH_USE_NT 1
#define _SMB_AUTH_USE_LM 1
```

3.2.3. Linux Samba Setup

Under Samba, the shared folders are managed via the `smb.conf` file. The `smb.conf` file is usually located in one of the following locations:

```
/usr/local/samba/lib/smb.conf
/usr/samba/lib/smb.conf
/etc/samba/smb.conf
/etc/smb.conf
```

The following is a simple portion of an `smb.conf` file that can be used to create a share for the Rabbit to be used in the samples. This portion assumes that a user named rabbit exists on the Linux machine.

```
[rabtest]
   comment = Rabbit Shared Folder
   path = /home/rabbit
   read only = no
   writable = yes
   public = no
   #force create mode = 0770
   #force directory mode = 0770
   valid users = rabbit, @users
```

All files and directories created by the rabbit will have permissions set only for the rabbit. Other users only have read permissions. The two commented out lines (force create mode/force directory mode) cause files created on the share to have read/write permissions set for the rabbit user and members of the same group. If you cannot delete files created by the rabbit you most likely have a permission problem and these two lines can help alleviate this.

Refer to your Linux documentation for further help locating, configuring, starting and stopping Samba.

3.2.4. Troubleshooting

If Thinshare cannot connect to a share on the server, try the following steps.

Create a new user account on the server that is sharing out the files. This should remove potential issues with user password authentication against a Domain Controller and the local server will perform the password authentication.
Ensure the LAN Manager authentication level is compatible with Thinshare.
Ensure SMB Signing is disabled.

In certain configurations, a Windows computer may be set by the domain controller or group policy to only use high encryption. Check the LAN Manager authentication level in group policy editor (Control Panel->Administrative Tools->Local Security Policy. Or, Start->Run 'gpedit.msc'):

Security Settings->Local Policies->Security Options->Network Security: LAN Manager authentication Level

Ensure that NT or NTLM keys are accepted and not set to 'Refuse'.

Make sure either LM or NTLM responses are being sent. The default setting in 2000 and XP is "Send LM & NTLM responses" or "Send NTLM response only" depending on the service pack.

Another setting that can impact connectivity is the following:

Security Settings->Local Policies->Security Options->Microsoft network server: Digitally sign communications (always)

This should be set to **disabled**.

Note: Make sure to run the "gpupdate" command after changing these settings to immediately apply the security policy.

3.3. Samples Synopsis

Several sample programs are included with ThinShare. The samples are located in the `\Samples\ThinShare` folder relative to the Dynamic C installation (or in `ThinShare\Samples` if the integrate with Dynamic C option was not selected during installation).

3.3.1. `hello_world_txt.c` - "Hello World.txt"

"Hello World.txt" is the closest to a hello world program for SMB you can probably get. This sample creates a new file on the server's SMB share and writes "Hello World!" into it.

Requirements: The user has create and write-access to the share.

3.3.2. `smbshell.c` - "ThinShell"

ThinShell provides an interactive command-line shell with a large number of available commands. Many common commands are available and virtually every aspect of the ThinShare SMB Library is utilized.

Requirements: Varies per command.

3.3.3. `cat.c` - "View File"

cat is named after the UNIX program of the same name. It is like the UNIX cat, or the DOS program type. It simply prints out the contents of a text file to the screen, in this case the Dynamic C stdio window.

Requirements: The input file exists on the server and the user account has read-access to the file.

3.3.4. `cp.c` - "Copy File"

cp is named after the UNIX program of the same name. It is like the UNIX cp or the DOS program copy. Two files are specified, the source and destination. The source file is copied into a new file named destination.

Requirements: The source file exists on the server and the user account has read-write access to the share.

3.3.5. `comprehensive.c` - "Comprehensive File I/O"

Comprehensive is a fairly comprehensive sample covering almost all of the filesystem based functions. This sample generates its own files and is thus self-sufficient. It starts by creating a directory and a file in that directory. The file is then the subject of the rest of the sample: it is written to, read from, renamed, etc.

Requirements: The user account must have create and write-access to the share.

3.3.6. `network_smb_dump.c` - "Server/Share Information Dump and Printout"

The Network SMB Dump sample generates a list of machines in the device's current workgroup. It then gets the list of shares on the server specified. This list can be written to the server as either a text file or as a file to be printed (a printer spool file). If the print option is attempted, the printer must support line-printer style print access (non-postscript printers such as many inkjet or dot-matrix printers work well).

Requirements: The workgroup must have a master browser that is accessible, the user account must have access to list shares on the server, and the user account must have access to create a print spool file on the printer (if printing mode is enabled).

Chapter 4. Getting Support

Standard technical support is over email. Other levels of support may be negotiated, to increase a support package contact <info@embeo.com>.

4.1. What to Provide When Contacting Support

- Dynamic C Version
- Processor / Development board (i.e., Rabbit 3000, RCM3200)
- Version of ThinShare
- ThinShare Configuration (What type of Node?)
- Possibly send a copy of your `smb_config.lib` file
- Server Configuration Windows/Linux (include OS version, Samba version, etc)
- Network Configuration (WINS servers?)

Provide a clear and detailed explanation of the problem experienced including what operations were executed.

4.2. Email Support

Send the information listed above to: <support@embeo.com> [mailto:support@embeo.com]. We try to respond within 24-48 hours during regular business hours, Monday thru Friday.

Chapter 5. License Agreement

By downloading, installing, copying, extracting, or otherwise accessing or using the EMBEO THIN-SHARE LIBRARY (the "SOFTWARE") you agree to all terms in this license agreement. If you do not agree to any or every part of this license agreement ("LICENSE") DO NOT INSTALL, USE, COPY, EXTRACT, OR ACCESS the SOFTWARE.

Embeo, Inc. grants the licensee a nonexclusive, nontransferable license to use and copy the SOFTWARE for usage by the licensee and those licensed within the licensee's representative business. The SOFTWARE may not be redistributed, sold, loaned, rented, leased, or re-sold except as a compiled binary inclusion in another piece of hardware by way of compilation and linking. The licensee may not create derivative works based on the SOFTWARE.

Embeo, Inc. disclaims all warranties, express or implied, that the software is fit for a particular purpose, accurate, or error-free. Embeo, Inc. does not guarantee any frequency of fixes, updates, or modifications to the SOFTWARE.

Embeo, Inc. will provide THE SOFTWARE in a binary and/or encrypted format to the licensee. All copies of the software are to remain within the licensed company.

The licensee may not directly or indirectly via a third party, decompile, reverse engineer, disassemble, or otherwise attempt to access the source code of the SOFTWARE. Nor may the licensee alter, modify, extract, derive, or reference, the source code in any way.

The licensee agrees not to hold Embeo, Inc. liable for any problems, monetary losses, or damages encountered through use, misuse, non-use, or inability to use the SOFTWARE.

Embeo, Inc. shall provide software updates to fix known bugs ("bugfixes"). These updates will occur on Embeo's schedule. This license does not guarantee any specific frequency of releases nor does it guarantee any discrete period between notification of a bug and the fix.

Embeo, Inc. shall provide technical support to the licensee, and only the licensee, via email and not to the licensee's customers or clients.

Part II. Technical Reference

Table of Contents

Chapter 6. Basic Configuration and Startup

ThinShare has both compile-time and run-time configuration options. This section covers both of these and SMB system initialization.

6.1. SMB Options

The following describes the basic compile time configuration options for the SMB system.

6.1.1. Host/Group Name

The Rabbit's host name and workgroup/domain name can be set at compile time using the following optional macros:

```
/*
 *  User macro to enable auto registration of SMB node name (Computer name)
 *    #define SMB_NODE_NAME "name"
 *
 *  User macro to enable auto registration of SMB group name (Workgroup/Domain)
 *    #define SMB_WORKGROUP_NAME "group"
 *
 */
```

These names can be a maximum of 15 characters and must not begin with a '*'. If the strings are longer, they will be truncated. These names can be set or changed at runtime by using the `smb_set_nodename` and `smb_set_wrkgrpname` functions.

6.1.2. Enabling the Browse Service

The Browse Service causes computer icons to appear in Network Neighborhood or My Network Places in Windows. It also allows searching for other machines and available shares. Since it is not required for proper SMB functionality, it is disabled by default. It can be enabled at compilation with the following macro:

```
/*
 * User macro to enable SMB Browse service (makes icon appear in network
 *  neighborhood).
 *    #define SMB_BROWSE_ENABLE<indexterm><primary>SMB_BROWSE_ENABLE</primary></in
 */
```

Note: In order to see an icon for the device in Network Neighborhood/My Network Places, there must be a master or backup browser for the workgroup (see Chapter 10, *Browse Service*). Also, it may take up to 12 minutes for the list to be refreshed.

6.2. System Startup

A call to `smb_init` will start the NetBIOS subsystem and then initialize the SMB layer. Below is a sample skeleton program:

```
/* Simple SMB Program */

#memmap xmem

#define SMB_NODE_NAME "RCM2200"
#define SMB_WORKGROUP_NAME "workgroup"
#define TCPCONFIG 3  /* ethernet with dhcp */
#use "dcrtcp.lib"
#use "smb.lib"

main()
{

   ifconfig( ... );  // Initialize the network interfaces

   sock_init();  // Startup the TCP/IP system

   if (smb_init() == _SMB_NOERRORS)  //Start Netbios & SMB
      printf("SMB system Init Success!\n");
   else printf("SMB system Init Failed.\n");

   //User SMB work here...

   smb_tick(); //Maintain health of the SMB system

   //Additional User SMB work here...
}
```

It is important to note the call to `smb_tick`. If the Browse Service is enabled, `smb_tick` must be called periodically to keep it running well. All SMB functions, however, automatically maintain the NetBIOS subsystem and contain a call to `tcp_tick`. This means that if the SMB subsystem is under frequent use, the software developer should not have to make explicit calls to `tcp_tick`. If there are times, however, when SMB functionality is not used for some time, then it is advisable to make calls to `smb_tick`, which also maintains the Netbios and TCP/IP systems. This insures that the ThinShare system runs smoothly. Once the SMB system is started, any calls to `tcp_tick` can be replaced by calls to `smb_tick` since it contains a call to `tcp_tick`.

Chapter 7. SMB Sessions

Before an SMB client can open any files on a given server, the client must open a session with the server. The `smb_connect` and `smb_disconnect` functions handle the work of opening and closing sessions with the desired SMB server. When a session is created, the user must supply the server name and the share name to access as well as a username and password if needed. An SMB Session ID (SID) is then returned and is used to identify the session in future SMB function calls. For example, when file operations such as read and write are performed.

```
/* Opening an SMB Session */

int sid;

//Create session on \\server\share
sid = smb_connect("server", "share", "username", "password");
if (sid < 0) {
   printf("Session Setup Failed.\n");
   return -1;
}
else printf("Session Setup Success!\n");

//Do some file operations...

//All done, close session
smb_disconnect(sid);
```

Once a session is opened by connecting to a share, any files or directories within that share may be accessed provided the user has appropriate access permissions. It is important to remember that the concept of current path or present working directory goes no further than the root of the share itself. This means that files must be opened using paths relative to the share. For example, to open the file: `\\server\share\directory\subdirectory\file.ext`, a session must first be opened to `\\server\share` followed by an open call on the file `directory\subdirectory\file.ext`.

Chapter 8. File Operations

Once a session is opened on a server, any number of file system operations can be performed. The API for file operations roughly mirrors the standard UNIX file I/O system call interface. The major difference being that a SID must be supplied as the first parameter to distinguish between open sessions when opening, renaming, or deleting files. Refer to Part III, “Function Reference” for more details about this.

ThinShare includes versions of the standard functions: `creat`, `open`, `close`, `read`, `write`, `lseek`, `stat`, `fstat`, `rename`, `unlink`, `chmod`, and `utime`. These functions are intuitive to use, having been designed to operate like their UNIX counterparts.

8.1. Opening and Closing Files

Opening files is done with `smb_creat` or `smb_open`. The standard `smb_open` can be used to open files for reading or writing, whereas `smb_creat` is mainly used to create files and therefore opens them for writing. If `smb_creat` is applied to an existing file, the file contents are truncated to zero bytes. These two functions return file descriptors used in future operations on the opened file. When access to the file is no longer needed, `smb_close` should be called to close the file on the server and de-allocate the file descriptor. There are a limited number (_SMB_FTAB_SIZE) of file descriptors within the ThinShare SMB file API and therefore, a maximum number of simultaneously opened files.

8.2. Reading And Writing to Files

Files can be read from or written to using the functions `smb_read` and `smb_write`. These functions require file descriptors (as returned by `smb_creat` or `smb_open`) to identify the file. The user must also supply a buffer for data transfer as well as the number of bytes to transfer.

As with the UNIX file I/O system call interface, file access is sequential. This means that successive reads or writes continue from the ending position in the file of the previous call. For example, suppose a new a file is created and two 100 byte `smb_write`'s are applied to it. At the end of the first write the file has 100 bytes of data. The second write will start at byte 101 and continue to write 100 more bytes resulting in a 200 byte file size. Internally, the file position is maintained as a pointer into the file data. Reads and writes to files always start at the current position and move the file pointer to reflect the operation performed. The `smb_lseek` function modifies the file pointer manually to enable jumps to any place in the file at any time. This can be useful to read the tail end of a file or return to the beginning of the file. Also if contiguous reads or writes are unneeded, `smb_lseek` can significantly reduce the overhead of unneeded network traffic and processor time.

8.3. Miscellaneous File Operations

Files can be renamed or moved with the `smb_rename` function. Because of the session boundaries, however, they cannot be moved from share to share with this function. Files can also be deleted by calling `smb_unlink`.

Two stat functions are available to get file status information such as size, attributes, and timestamps. If the file is currently opened, `smb_fstat` can be called with the file descriptor to get information about the file. If the file is not open, `smb_stat` will return the status information by supplying the file name.

Below is a simple example of file access:

```
#define DATABUFSIZ 256
#define FILENAME "textfile.txt"

//Simple file access

char buffer[DATABUFSIZ+1];  //buffer to contain file data
int fd;  //file descriptor

//Created server session

//now open file with read permissions
fd = smb_open(sid, FILENAME, SMB_O_RDONLY);
printf("Target file opened...\n");

//read first DATABUFSIZ bytes of file
buffer[DATABUFSIZ] = 0;  //NULL terminate
smb_read(fd, buffer, DATABUFSIZ);
printf("First %d characters of %s:\n%s\n", DATABUFSIZ, FILENAME, buffer);

//close file
smb_close(fd);
```

To change and save file attributes and timestamps back to the server, the functions `smb_chmod` and `smb_utime` should be used, respectively. These functions take a regular filename versus a file descriptor. Refer to Part III, “Function Reference” for more information regarding the use of these functions.

Chapter 9. Filesystem Directory Operations

Connecting to an SMB share is similar to opening an ftp connection or a shell to a server. You can perform operations on files and they are organized into a directory tree, etc. However, there is no concept of a 'current working directory' (DOS **cd** command, UNIX **pwd** command) as far as this API is concerned. Therefore, there are no change directory operations (such as cd/chdir). Files are referenced with their entire path relative to the share connected via the `smb_connect` function call.

Example, with the following path:

```
\\Server\Share\Directory\File.ext
```

The connected share on 'Server' is named 'Share'. The file File.ext must always be referred to by the pathname `Directory\File.ext` versus just `File.ext`.

9.1. Creating and Deleting Directories

Even without the ability to change directories from within an SMB share, they can still be created and deleted with the directory functions `smb_mkdir` and `smb_rmdir`. The paths specified to these functions must also be relative to the root of the share they are in, as with file names.

9.2. Searching Directories

Directory searches can be performed in any path in a share using the search functions `smb_sopen` and `smb_sread`. The search is first opened with `smb_sopen`, then the returned names are retrieved with one or more calls to `smb_sread`. Directory searches support standard wildcards in the path string specified in `smb_sopen`. The files in the share root, for example, can be listed by specifying `'*'` as the search string, while `'\path\*'` will list files in the 'path' subdirectory. Search results can also be filtered by file attributes such as listing directories only.

To keep overhead low for this functionality, the user must supply the needed data structure to keep the search information as well as a character buffer to use as a temporary cache for returned filenames. The size of the buffer should be larger than the number of characters of the longest expected filename. A buffer size of 50 bytes or greater should suffice for standard filenames.

When search results are no longer needed, a call to `smb_sclose` should be made to reset the search data structure and release the user's buffer space. Please refer to Part III, “Function Reference” for more information about the directory functions.

Chapter 10. Browse Service

The SMB Browse Service is an advertising mechanism that allows SMB clients and servers to make their presence known on the local network. It is what makes the Windows Network Neighborhood/My Network Places possible. Using the Browse Service, SMB clients and servers announce themselves to adjacent machines on the network, as well as provide information to the user about themselves such as available resources.

To enable Browse Service functionality, the SMB_BROWSE_ENABLE macro must be defined, either in the SMB configuration file or in the user's source file. When the Browse Service is enabled, the SMB subsystem will automatically announce itself to others on the network via calls to `smb_tick`. It is because of these periodic announcements that the icons appear within the Network Neighborhood.

10.1. Network and Server Operations

An SMB client must know the name of an SMB server before it can open a session with the server. The `smb_list_servers` function returns a list of servers in the client's workgroup or domain. Within every workgroup/domain there is a 'Master Browser' that maintains a list of servers available within the group, and `smb_list_servers` gets the list from this server. If the browser machines are secure, meaning that they do not accept anonymous connections, the function `smb_list_servers_ex` can be used to get the list. It is possible that there is no Master Browser, which is the case if the Rabbit device is the only member of the group. Since this SMB suite does not support running as a browser, the Rabbit device would not be seen from adjacent machines from other workgroups or domains if it is the only member of its workgroup. The function `smb_list_servers_manual` can be used in which the user specifies a server from another workgroup to query for a list of available servers. To search for all available servers on every workgroups/domain the command `smb_list_all_servers` can be used. This function queries all of the Master Browsers on the network for their list of available servers and concatenates the responses into a single list.

Note: `smb_list_servers` and related functions always return a list of servers on the queried server's workgroup/domain.

Once you have a server to connect to, you need to know what shares are available. The function `smb_list_shares` has a similar interface to `smb_list_servers`, but instead of returning a list of servers, it returns a list of shares on the specified server. Both of these functions require a user-supplied buffer for storing the returned data. It is important to note, again, that these functions are only available when the Browse Service is enabled. Refer to Part III, “Function Reference” for further information regarding the use of `smb_list_servers` and `smb_list_shares`.

10.2. Printer Support

The SMB subsystem can create network printer spool files, providing basic line printer support. No specific printer drivers are included and all print data is assumed to be in raw format, so preformatted data must be supplied to take advantage of specific printer features beyond basic plain text.

To print a document, open an SMB session to a shared network printer using the `smb_connect` function. Then call `smb_lpt_open` to create a printer spool file. The open function will return a file descriptor to be used with subsequent calls to `smb_write` accompanied by the print data. Once the data is written to the file, a call to the `smb_lpt_close` function will close the file and send the data to the printer. Finally, `smb_disconnect` can be called if there are no more documents to be printed.

For basic plain text documents, the printer may not automatically wrap lines of text, so newline CRLF control sequences (`\r\n`) should be included in the print data in the appropriate locations. Further, the last

three data characters should be `\r\n\f`, to signal the end of the print document. If these three characters do not conclude the print data, the printer may indicate an error status or it may wait for further print data. For an example using the printer, refer to the sample program `network_smb_dump.c`.

Chapter 11. Advanced System Configuration

ThinShare provides many macros for customizing NetBIOS and SMB. They are discussed in the following subsections.

11.1. NetBIOS Options

The following describes the various compile time configuration options for the underlying NetBIOS subsystem.

11.1.1. Node Type

The NetBIOS system can operate in one of three node types: B, P, or M. B-nodes operate in the absence of a WINS (NBNS) server, whereas P-node or M-nodes use WINS for name registration and resolution. The operational differences between these node types is beyond the scope of this document, however, they can be found in RFC1001. The node type may be configured statically at compile-time or dynamically at run-time. For static node type configuration, one of the following macros may be defined:

```
/*  User configurable node-type macros (define one only)
*
*  NETBIOS_B_NODE   - B-node support only: broadcast node (smallest code)
*  NETBIOS_P_NODE   - P-node support only: point-to-point node
*  NETBIOS_M_NODE   - M-node support only: mixed-mode node; mixed B and P
*  NETBIOS_ALL_NODE - Dynamic Support of all Nodes (runtime configuration)
*
*    Using B or P may significantly reduce object code size.
*/
```

If the node type is not defined at compile-time, B-node configuration will be used. If dynamic mode is selected, the node type is configured automatically depending on the local IP address and WINS IP address, if used.

11.1.2. WINS Server

The default WINS server is specified by the following macro:

```
/*  User macro to setup the default NBNS (M/P nodes only):
 *    #define SMB_WINS_SERVER "[dotted IP string]"
 */
```

It can be set or changed at runtime using the `smb_set_WINS` function.

11.1.3. DHCP Support (Dynamic C 8.30 and Later)

The NetBIOS system's WINS server can also be configured over DHCP (i.e. the DHCP Server fills in DHCP Option 44). The following example illustrates the setup of DHCP WINS Support.

```
/*
 * DHCP Callback for WINS/NBNS server: SMB_DHCP_CALLBACK
 *
 * Example:
 *
 *   char dhcp_opts[];
 *
 *   dhcp_opts[] = {DHCP_VN_NBNS}; //DHCP option for WINS/NBNS server
 *
 *   ipconfig(IF_ETH0,
 *            IFS_DHCP,1,
 *            IFS_DHCP_OPTIONS, 1, &dhcp_opts, SMB_DHCP_CALLBACK,
 *            IFS_END); //sets dhcp system to call SMB_DHCP_CALLBACK for WINS
 */
```

If WINS server information is acquired by the use of DHCP, Dynamic C 8.30 or later must be used. However, if WINS is not used, the Dynamic C version 7.05 and later DHCP system is supported.

11.2. SMB Configuration File and Macros

There are many configuration tuning parameters that can be utilized to tailor the performance of ThinShare as desired for any application. They will not be discussed here, but the configuration file `smb_config.lib` contains the complete collection as well as the basic options discussed in this document.

All configuration options are listed in the standard `smb_config.lib`. Each option in the file is documented as to its purpose. To use `smb_config.lib`, simply '#use' it before `smb.lib`. Any option in `smb_config.lib` that is commented out will be set to its default value (defaults are specified in the file). To use an option, simply uncomment the line and change the value to the desired setting. It is important to note that any options defined within `smb_config.lib` must not also be specified in the main program's .c file or compilation errors will result due to duplicate macro definitions. It is easiest just to define all the options in one place.

Using `smb_config.lib`, it is possible to create multiple custom configuration files and use different configurations per project. They must also be listed in the `LIB.DIR` file as the Dynamic C compiler requires. Below is a listing of a few of the many ThinShare configuration options available:

Node Name
Workgroup/Domain Name
WINS IP Address
Node Type
Browse Service Enable/Disable
Security Key Types
Max Number of Open Files
Max Number of Concurrent Sessions
Various Internal Buffer Sizes
Host Resolution Name Cache Size

Refer to `smb_config.lib` for a full description of the available configuration options.

Part III. Function Reference

Name

smb_chmod

Synopsis

```
int smb_chmod(int sid, char *name, uint16 st_attributes);
```

Description

```
chmod sets the file's attributes to 'st_attributes'.

NOTE: Not all attributes are supported on all SMB servers.
The archive and readonly attributes are the most widely
supported.  In some cases the attributes can be read but not
set. (SAMBA under Linux and many NAS drives do not fully support
attributes).
```

Parameters

```
PARAMETER1: SMB session id
PARAMETER2: filename to set the attributes on
PARAMETER3: new attributes

            st_attributes can be any number of the following bit flags OR'd
            together:

            #define _SMB_FSIO_READONLY    0x0001
            #define _SMB_FSIO_HIDDEN      0x0002
            #define _SMB_FSIO_SYSTEM      0x0004
            #define _SMB_FSIO_ARCHIVE     0x0020
```

Return Value

```
0 on success.

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - filename null or too long
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

`smb_utime`, `smb_fstat`, `smb_stat`

Name

smb_close

Synopsis

```
int smb_close(int fd);
```

Description

```
Closes open file descriptor 'fd' and returns the resources used
to the open file table.
```

Parameters

```
PARAMETER1: file descriptor
```

Return Value

```
0 on success

on error (<0):

_SMB_ERR_INVAL_FD        - invalid file descriptor
_SMB_ERR_TIMEOUT         - fatal timeout waiting for response
_SMB_ERR_TRANSPORT       - fatal NetBIOS error in send/recv
```

See Also

`smb_write`, `smb_read`, `smb_open`

Name

smb_connect

Synopsis

```
int smb_connect(char *server, char *share,
                char *user, char *passwd);
```

Description

```
Opens an SMB session to a server on a given path.  Username is
authenticated with NT and/or NTLM session keys generated
from password depending on _SMB_AUTH_USE_NT / _SMB_AUTH_USE_LM.
Return value on success is a session id used in other smb
commands.  All fields (except passwd) are case insensitive.

To disable NT/NTLM or LM keys define the respective macro to
0, likewise to use it define it to 1 (default).

NT/NTLM - _SMB_AUTH_USE_NT
LM      - _SMB_AUTH_USE_LM

EXAMPLE:       To connect to \\SERVER\SHARE:
sid = smb_connect("server", "share", "user", "password");
```

Parameters

```
PARAMETER1: server's NetBIOS name (no backslashes) (max length 15,
            truncated if longer)

PARAMETER2: share on server (\SHARE or SHARE) (max length
            _SMB_MAXSHARE, default: 80)

PARAMETER3: username (max length _SMB_MAXUN, default: 20)

PARAMETER4: password (max length _SMB_MAXPW, default: 128)
```

Return Value

```
on success (>=0): smb session id (sid)

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - Share/server/user/passwd too long
                            or server/share null
_SMB_ERR_OUT_OF_SID - Out of SMB sessions/too many connections
_SMB_ERR_HOST_NOT_FOUND - Server not found/did not respond
_SMB_ERR_ACCESS_DENIED - Bad user/password or access permissions
_SMB_ERR_INVAL_PATH - Invalid share format
```

```
_SMB_ERR_INVAL_SHARE - Share name is invalid/missing
_SMB_ERR_CALL - Could not establish connection with Server
_SMB_ERR_UNSUPPORTED_DIALECT - Incompatable with Server
_SMB_ERR_UNSUPPORTED_FLAG - Incompatable with Server
_SMB_ERR_TRANSPORT - Fatal error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response. Fatal error.
```

See Also

smb_disconnect, smb_open, smb_sopen

Name

smb_convutime

Synopsis

```
void smb_convutime(smb_utime_t utime, smb_datetime *dt);
```

Description

```
Calculates current date/time info from a given unix time 'utime'
UNIX time is based on the number of seconds since
Jan 1, 1970 00:00:00.0 UTC

smb_convutime takes into account leap year (every four years
starting 2/29/1972). Assumes no dates before 1972 to simplify
calculations.

typedef struct {
        uint16 sec;
        uint16 min;
        uint16 hour;    // 24 hour format
        uint16 day;     // day of month
        uint16 month;   // 1=Jan, 2=Feb, etc
        uint16 year;    // four digit digit format, ie 2003
} smb_datetime;

Time format: hour:min:sec
Date format: month/day/year
```

Parameters

```
PARAMETER1: input time in UNIX time format
PARAMETER2: pointer to smb_datetime struct to save date/time info into
```

Return Value

```
none.
```

See Also

```
smb_mkutime, smb_fstat, smb_stat
```

Name

smb_creat

Synopsis

```
int smb_creat(int sid, char *name);
```

Description

```
creat creates a new file named 'name' on the SMB session 'sid'.
If the file exists, creat will truncate the file to zero bytes.
Returns a file descriptor to be used with other SMB file I/O
functions such as smb_write, smb_read, and smb_close.
```

Parameters

```
PARAMETER1: SMB session descriptor
PARAMETER2: filename of file to create
```

Return Value

```
on success (>=0):

file descriptor to be used in calls to smb_read, smb_write, and
smb_close.

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - bad filename
_SMB_ERR_NO_RESOURCES     - out of file descriptors
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

smb_write, smb_read, smb_close

Name

smb_datetime

Synopsis

```
typedef struct _smb_datetime {
        uint16 sec;     // seconds
        uint16 min;     // minutes
        uint16 hour;    // 24 hour format
        uint16 day;     // day of the month (starting with 1)
        uint16 month;   // 1=Jan, 2=Feb, etc
        uint16 year;    // four digit year, ie 2004
} smb_datetime;
```

Description

Data structure for use with smb_mkutime and smb_convutime functions.

See Also

smb_mkutime, smb_convutime

Name

smb_dir_info

Synopsis

```
typedef struct _smb_dir_info {
        char count;    //number of valid filename pointers
        char *filename[_SMB_DIR_SRCH_SIZE]; //array of filenames
} smb_dir_info;
```

Description

Data structure for use with the smb_sread() directory searching function. It is returned via a user supplied address to a 'smb_dir_info' pointer {&(smb_dir_info *)}.

'count' is the number of valid filenames in the 'filename' array.

A fixed maximum number of filenames will be returned, set by the _SMB_DIR_SRCH_SIZE configuration parameter. Fewer may be returned if the names are lengthy or if they are the last of the search results.

See Also

smb_sread, smb_sopen, smb_sclose

Name

smb_disconnect

Synopsis

```
int smb_disconnect(int sid);
```

Description

```
closes an open SMB session
```

Parameters

```
PARAMETER1: session id of open session to close
```

Return Value

```
on success: _SMB_NOERRORS

on error (<0):

_SMB_NOERRORS
_SMB_ERR_INVAL_SID - SID is invalid

In the following cases the session is locally disconnected
and all resources are deallocated, despite errors:

_SMB_ERR_TRANSPORT - Error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response.
```

See Also

```
smb_connect
```

Name

smb_fchmod

Synopsis

```
int smb_fchmod(int fid, uint16 st_attributes);
```

Description

```
fchmod sets the file's attributes to 'st_attributes'.
```

Parameters

```
PARAMETER1: file descriptor
PARAMETER2: new attributes

            st_attributes can be any number of the following bit flags OR'd
            together:

            #define _SMB_FSIO_READONLY    0x0001
            #define _SMB_FSIO_HIDDEN      0x0002
            #define _SMB_FSIO_SYSTEM      0x0004
            #define _SMB_FSIO_VOLUME      0x0008
            #define _SMB_FSIO_DIRECTORY   0x0010
            #define _SMB_FSIO_ARCHIVE     0x0020
```

Return Value

```
0 on success, < 0 on error.
```

See Also

smb_utime, smb_fstat, smb_stat, smb_chmod

Name

smb_fstat

Synopsis

```
int smb_fstat(int fd, struct smb_stat_t *stat);
```

Description

```
fstat gets information about an open file on file descriptor
'fd' and saves it in the user-supplied smb_stat_t structure.

The smb_stat_t structure contains the file's size, attributes,
and modification time.

struct smb_stat_t {
        uint32       st_size;       // total size in bytes
        uint16       st_attributes; // DOS attributes
        smb_utime_t  st_time;       // time of last data modification
};

The modification time, st_time, can be converted to useful
terms via the smb_convutime function.

NOTE: The time returned by smb_fstat has a 2 second resolution
and thus may be off compared to the time returned to smb_stat.
```

Parameters

```
PARAMETER1: file descriptor
PARAMETER2: pointer to smb_stat_t structure

            st_attributes can be any number of the following bit flags OR'd
            together:

            #define _SMB_FSIO_READONLY    0x0001
            #define _SMB_FSIO_HIDDEN      0x0002
            #define _SMB_FSIO_SYSTEM      0x0004
            #define _SMB_FSIO_VOLUME      0x0008
            #define _SMB_FSIO_DIRECTORY   0x0010
            #define _SMB_FSIO_ARCHIVE     0x0020
```

Return Value

```
_SMB_NOERRORS (0) on success.

on error (<0):

_SMB_ERR_INVAL_FD        - invalid file descriptor
```

```
_SMB_ERR_ACCESS_DENIED   - bad file access permissions
_SMB_ERR_INVAL_SHARE     - bad sharing mode on file/server err
_SMB_ERR_INVAL_FID       - internal data error
_SMB_ERR_INVAL_TID       - internal data error
_SMB_ERR_INVAL_UID       - internal data error
_SMB_ERR_MALFORMED       - server success but malformed smb
_SMB_ERR_TIMEOUT         - fatal timeout waiting for response
_SMB_ERR_TRANSPORT       - fatal NetBIOS error in send/recv
```

See Also

`smb_stat`, `smb_convutime`

Name

smb_get_comment

Synopsis

```
char *smb_get_comment();
```

Description

Gets the browse comment.

SMB_BROWSE_ENABLE must be defined to use this function.

Return Value

Browse comment

See Also

smb_set_comment

Name

smb_get_nodename

Synopsis

```
char *smb_get_nodename(void);
```

Description

Returns the node name.

Return Value

Pointer to string containing node name.

See Also

smb_set_nodename

Name

smb_get_nodetype

Synopsis

```
uint8 smb_get_nodetype();
```

Description

Gets the NetBIOS node type.

Return Value

Node type which is one of the following:

```
SMB_BNODE = Broadcast node
SMB_MNODE = Mixed-mode node
SMB_PNODE = Point-to-point node
```

See Also

smb_set_nodetype

Name

smb_get_WINS

Synopsis

```
uint16 smb_get_WINS(uint32 *wins, uint16 count);
```

Description

Returns 'count' items from the NBNS name table.

Parameters

PARAMETER1: An array large enough to hold 'count' NBNS server IP addresses

PARAMETER2: Max number IPs to return.

Return Value

Number of NBNS servers returned.

See Also

smb_set_WINS

Name

smb_get_wrkgrpname

Synopsis

```
char *smb_get_wrkgrpname(void);
```

Description

Returns the workgroup name.

Return Value

Pointer to string containing workgroup name.

See Also

smb_set_wrkgrpname

Name

smb_init

Synopsis

```
int smb_init(void);
```

Description

```
Starts the smb/cifs client system (and netbios). If name macros
SMB_NODE_NAME or SMB_WORKGROUP_NAME are defined, the names will
be registered automatically.

smb_init must be called before other SMB functions.
```

Return Value

```
_SMB_NOERRORS              - System started normally.
_SMB_ERR_NAME_ASSIGNMENT   - A defined name SMB_NODE_NAME or
                             SMB_WORKGROUP_NAME could not be
                             registered.
```

Name

smb_list_all_servers

Synopsis

```
int smb_list_all_servers(smb_server_info *srvinf, void *buffer,
                         int bufsize, char *user, char *pass);
```

Description

```
Gets a list of all the machines on the network.  Servers list
is generated from all responding local master browsers. Only
machine names of those machines in workgroups that have master
browsers will be returned.

This function will block for _SMB_MASTER_BROWSE_WAIT
milliseconds (default 50ms) waiting for master browse servers
to respond.  If there is no response or minimal responses,
increase the value of _SMB_MASTER_BROWSE_WAIT via a define or
call this function multiple times.

SMB_BROWSE_ENABLE must be defined to use this function.

In some network configurations it is necessary to supply a
valid username/password combination.

Otherwise to specify an anonymous connection, call as follows:

smb_list_all_servers(&srvinf, &recvbuf, RECVBUFSIZ, "\0", "\0");

Alternatively the username "guest" may be attempted.
```

Parameters

```
PARAMETER1: Pointer to valid smb_server_info structure to contain returned
            server count and list.

            typedef struct {
                int  count;  //number of server strings in name array
                int  total;  //total number of servers available
                char **name; //char *name[count] - list of server names
            } smb_server_info;

PARAMETER2: User-allocated memory that will contain list of returned server
            names.
PARAMETER3: Size of buffer in bytes.
PARAMETER4: Username for server connection (for anonymous use NULL)
PARAMETER5: Password for server connection (for anonymous use NULL)
```

Return Value

```
on success: _SMB_NOERRORS

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - null srvinf or buffer param
_SMB_ERR_NO_MSBROWSE - no local master browser servers found
```

See Also

smb_list_servers, smb_list_servers_ex, smb_server_info

Name

smb_list_servers

Synopsis

```
int smb_list_servers(smb_server_info *srvinf, void *buffer,
                     int bufsize);
```

Description

```
Gets a list of machines on the network in the current workgroup.
Queries the workgroup/domain's master browser for the list of
servers in the workgroup.

This function can block for a maximum of _SMB_BU_LIST_MAX_WAIT
milliseconds (default 5000ms).

SMB_BROWSE_ENABLE must be defined to use this function.

Some servers will not allow anonymous connections, and the
smb_list_servers function uses an anonymous connection.  To
connect to these servers use the smb_list_servers_ex() function:

int smb_list_servers_ex(smb_server_info *srvinf, void *buffer,
int bufsize, char *user, char *password);
```

Parameters

```
PARAMETER1: Pointer to valid smb_server_info structure to contain returned
            server count and list.

            typedef struct {
                int  count;  //number of server strings in name array
                int  total;  //total number of servers available
                char **name; //char *name[count] - list of server names
            } smb_server_info;

PARAMETER2: User-allocated memory that will contain list of returned server
            names.
PARAMETER3: Size of buffer in bytes.
```

Return Value

```
on success:

_SMB_NOERRORS
_SMB_MOREDATA - more servers available than can fit in SMB

on error (<0):
```

```
_SMB_ERR_INVAL_PARAMETERS - null srvinf or buffer param
_SMB_ERR_BU_TIMEOUT - backup server didn't respond in time
_SMB_ERR_SERVICE_NOT_RUNNING - server not a master browser
_SMB_ERR_NO_RESOURCES - server out of connections/not master
_SMB_ERR_UNKNOWN - server most likely not a master browser
_SMB_ERR_PARAMCOUNT - Incompatable with server
_SMB_ERR_WORDCOUNT - Incompatable with server
_SMB_ERR_INVAL_PARAMETERS - User/passwd too long
_SMB_ERR_OUT_OF_SID - Out of SMB sessions/too many connections
_SMB_ERR_HOST_NOT_FOUND - Server not found/did not respond
_SMB_ERR_ACCESS_DENIED - Bad user/password or access permissions
_SMB_ERR_INVAL_PATH - Invalid share format
_SMB_ERR_INVAL_SHARE - Share name is invalid/missing
_SMB_ERR_CALL - Could not establish Connection with server
_SMB_ERR_UNSUPPORTED_DIALECT - Incompatable with server
_SMB_ERR_UNSUPPORTED_FLAG - Incompatable with server
_SMB_ERR_TRANSPORT - Fatal error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response. Fatal error.
```

See Also

smb_list_servers_manual, smb_server_info, smb_list_servers_ex

Name

smb_list_servers_ex

Synopsis

```
int smb_list_servers_ex(smb_server_info *srvinf, void *buffer,
                        int bufsize, char *user, char *pass);
```

Description

```
Gets a list of machines on the network in the current workgroup.
Queries the workgroup/domain's master browser for the list of
servers in the workgroup.

This function can block for a maximum of _SMB_BU_LIST_MAX_WAIT
milliseconds (default 5000ms).

SMB_BROWSE_ENABLE must be defined to use this function.
```

Parameters

```
PARAMETER1: Pointer to valid smb_server_info structure to contain returned
            server count and list.

            typedef struct {
                int  count;  //number of server strings in name array
                int  total;  //total number of servers available
                char **name; //char *name[count] - list of server names
            } smb_server_info;

PARAMETER2: User-allocated memory that will contain list of returned server
            names.
PARAMETER3: Size of buffer in bytes.
PARAMETER4: Username for server connection
PARAMETER5: Password for server connection
```

Return Value

```
on success:

_SMB_NOERRORS
_SMB_MOREDATA - more servers available than can fit in SMB

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - null srvinf or buffer param
_SMB_ERR_BU_TIMEOUT - backup server didn't respond in time
_SMB_ERR_SERVICE_NOT_RUNNING - server not a master browser
_SMB_ERR_NO_RESOURCES - server out of connections/not master
```

```
_SMB_ERR_UNKNOWN - server most likely not a master browser
_SMB_ERR_PARAMCOUNT - Incompatable with server
_SMB_ERR_WORDCOUNT - Incompatable with server
_SMB_ERR_INVAL_PARAMETERS - User/passwd too long
_SMB_ERR_OUT_OF_SID - Out of SMB sessions/too many connections
_SMB_ERR_HOST_NOT_FOUND - Server not found/did not respond
_SMB_ERR_ACCESS_DENIED - Bad user/password or access permissions
_SMB_ERR_INVAL_PATH - Invalid share format
_SMB_ERR_INVAL_SHARE - Share name is invalid/missing
_SMB_ERR_CALL - Could not establish Connection with server
_SMB_ERR_UNSUPPORTED_DIALECT - Incompatable with server
_SMB_ERR_UNSUPPORTED_FLAG - Incompatable with server
_SMB_ERR_TRANSPORT - Fatal error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response. Fatal error.
```

See Also

`smb_list_servers_manual`, `smb_server_info`, `smb_list_servers`

Name

smb_list_servers_manual

Synopsis

```
int smb_list_servers_manual(char *server,
                            smb_server_info *srvinf,
                            void *buffer, int bufsize,
                            char *user, char *pass);
```

Description

Gets a list of machines on the network in the workgroup of the named server. Queries 'server' for the list of servers in its workgroup.

This is a manual version of smb_list_servers which allows querying any server for its server list. Not all servers keep this list, and they will respond with no list or an error.

Most servers use anonymous connections, call as follows:

```
smb_list_servers_manual(server, &srvinf, &recvbuf, RECVBUFSIZ,
"\0", "\0");
```

Some servers will not allow anonymous connections. To connect to these servers specify a valid username/password.

SMB_BROWSE_ENABLE must be defined to use this function.

Parameters

PARAMETER1: NetBIOS name of server to query for server list (max length 15)
PARAMETER2: Pointer to valid smb_server_info structure to contain returned server count and list.

```
typedef struct {
    int  count;  //number of server strings in name array
    int  total;  //total number of servers available
    char **name; //char *name[count] - list of server names
} smb_server_info;
```

PARAMETER3: User-allocated memory that will contain list of returned server names.
PARAMETER4: Size of buffer in bytes.
PARAMETER5: Username for server connection (for anonymous use NULL)
PARAMETER6: Password for server connection (for anonymous use NULL)

Return Value

```
on success:

_SMB_NOERRORS
_SMB_MOREDATA - more servers available than can fit in SMB

on error (<0):

_SMB_ERR_SERVICE_NOT_RUNNING - server not a master browser
_SMB_ERR_NO_RESOURCES - server out of connections/not master
_SMB_ERR_UNKNOWN - server most likely not a master browser
_SMB_ERR_PARAMCOUNT - Incompatable with server
_SMB_ERR_WORDCOUNT - Incompatable with server
_SMB_ERR_INVAL_PARAMETERS - Server/user/passwd too long or null
                            srvinf/buffer params
_SMB_ERR_OUT_OF_SID - Out of SMB sessions/too many connections
_SMB_ERR_HOST_NOT_FOUND - Server not found/did not respond
_SMB_ERR_ACCESS_DENIED - Bad user/password or access permissions
_SMB_ERR_INVAL_PATH - Invalid share format
_SMB_ERR_INVAL_SHARE - Share name is invalid/missing
_SMB_ERR_CALL - Could not establish Connection with server
_SMB_ERR_UNSUPPORTED_DIALECT - Incompatable with server
_SMB_ERR_UNSUPPORTED_FLAG - Incompatable with server
_SMB_ERR_TRANSPORT - Fatal error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response. Fatal error.
```

See Also

smb_list_servers, smb_list_servers_ex, smb_server_info

Name

smb_list_shares

Synopsis

```
int smb_list_shares(char *server, smb_share_info *shrinf,
                    void *buffer, int bufsize);
```

Description

```
Gets a list of available shares on the named server.

Some servers will not allow anonymous connections, and the
smb_list_shares function uses an anonymous connection.  To
connect to these servers use the smb_list_shares_ex() function:

int smb_list_shares_ex(char *server, smb_share_info *shrinf,
void *buffer, int bufsize, char *user, char *pass);

SMB_BROWSE_ENABLE must be defined to use this function.
```

Parameters

```
PARAMETER1: NetBIOS name of server to query for share list.
PARAMETER2: Pointer to valid smb_share_info structure to contain returned
            share count and list.

            typedef struct {
                int  count;  //number of shares returned
                int  total;  //total number of shares available
                char **name; //char *name[count] - list of server names
            } smb_server_info;

PARAMETER3: User-allocated memory that will contain list of returned shares
PARAMETER4: Size of buffer in bytes
```

Return Value

```
on success: _SMB_NOERRORS

on error (<0):

_SMB_ERR_SERVICE_NOT_RUNNING - Server not running
_SMB_ERR_ACCESS_DENIED - Bad user/password or access permissions
_SMB_ERR_NO_RESOURCES - Server out of connections/not master
_SMB_ERR_INVAL_PARAMETERS - Server name too long or null
                            shrinf or buffer params
_SMB_ERR_OUT_OF_SID - Out of SMB sessions/too many connections
_SMB_ERR_HOST_NOT_FOUND - Server not found/did not respond
```

```
_SMB_ERR_INVAL_PATH - Bad server name format/not a server
_SMB_ERR_INVAL_SHARE - Server not running as a server
_SMB_ERR_CALL - Could not establish connection with server
_SMB_ERR_UNSUPPORTED_DIALECT - Incompatable with server
_SMB_ERR_UNSUPPORTED_FLAG - Incompatable with server
_SMB_ERR_UNKNOWN - Incompatible with server or server error
_SMB_ERR_PARAMCOUNT - Incompatable with server
_SMB_ERR_WORDCOUNT - Incompatable with server
_SMB_ERR_TRANSPORT - Fatal error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response. Fatal error.
```

See Also

smb_list_shares_ex, smb_share_info

Name

smb_list_shares_ex

Synopsis

```
int smb_list_shares_ex(char *server, smb_share_info *shrinf,
                       void *buffer, int bufsize, char *user, char *pass);
```

Description

```
Gets a list of available shares on the named server.
```

Parameters

```
PARAMETER1: NetBIOS name of server to query for share list.
PARAMETER2: Pointer to valid smb_share_info structure to contain returned
            share count and list.

            typedef struct {
                int  count;  //number of shares returned
                int  total;  //total number of shares available
                char **name; //char *name[count] - list of server names
            } smb_server_info;

            SMB_BROWSE_ENABLE must be defined to use this function.

PARAMETER3: User-allocated memory that will contain list of returned shares
PARAMETER4: Size of buffer in bytes
PARAMETER5: Username for server connection
PARAMETER6: Password for server connection
```

Return Value

```
on success: _SMB_NOERRORS

on error (<0):

_SMB_ERR_SERVICE_NOT_RUNNING - Server not running
_SMB_ERR_ACCESS_DENIED - Bad user/password or access permissions
_SMB_ERR_NO_RESOURCES - Server out of connections/not master
_SMB_ERR_INVAL_PARAMETERS - Server/user/passwd too long or null
                            shrinf or buffer params
_SMB_ERR_OUT_OF_SID - Out of SMB sessions/too many connections
_SMB_ERR_HOST_NOT_FOUND - Server not found/did not respond
_SMB_ERR_INVAL_PATH - Bad server name format/not a server
_SMB_ERR_INVAL_SHARE - Server not running as a server
_SMB_ERR_CALL - Could not establish connection with server
_SMB_ERR_UNSUPPORTED_DIALECT - Incompatable with server
_SMB_ERR_UNSUPPORTED_FLAG - Incompatable with server
```

```
_SMB_ERR_UNKNOWN - Incompatible with server or server error
_SMB_ERR_PARAMCOUNT - Incompatable with server
_SMB_ERR_WORDCOUNT - Incompatable with server
_SMB_ERR_TRANSPORT - Fatal error in NetBIOS layer send/recv
_SMB_ERR_TIMEOUT - Timed out waiting for response. Fatal error.
```

See Also

smb_list_shares, smb_share_info

Name

smb_lpt_close

Synopsis

```
int smb_lpt_close(int fd);
```

Description

```
Closes open printer spool file descriptor 'fd' and returns the
resources to the open file table.
```

Parameters

```
PARAMETER1: file descriptor
```

Return Value

```
0 on success

on error (<0):

_SMB_ERR_INVAL_FD        - invalid file descriptor
_SMB_ERR_TIMEOUT         - fatal timeout waiting for response
_SMB_ERR_TRANSPORT       - fatal NetBIOS error in send/recv
```

See Also

`smb_write`, `smb_lpt_open`

Name

smb_lpt_open

Synopsis

```
int smb_lpt_open(int sid, char *name, int mode);
```

Description

```
Open a print spool file named 'name' over SMB session 'sid'.
Actual spool filename is generated by server by adding random
characters to end of 'name'.

NOTES about printing:

1. Printer may not auto wrap text, user must keep track of line
width and insert newline control sequences ("\r\n") manually.

2. Complete printout with "\r\n\f" (at the end of last write)
to cause printer to finish with current page.

These two strings have the following defined names:

#define   SMB_LPT_NEWLINE     "\r\n"
#define   SMB_LPT_END         "\r\n\f"
```

Parameters

```
PARAMETER1: smb session id
PARAMETER2: printer spool file to create
PARAMETER3: printing mode, text or graphics depending on:

            SMB_LPT_TEXT_MODE 0
            SMB_LPT_GRFX_MODE 1
```

Return Value

```
on success (>=0):

file descriptor to be used in calls to smb_write, smb_lpt_close

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - spool filename null or too long, or
                            invalid mode
_SMB_ERR_NO_RESOURCES     - out of file descriptors
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
```

```
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

`smb_write`, `smb_lpt_close`

Name

smb_lseek

Synopsis

```
uint32 smb_lseek(int fd, int32 offset, int origin);
```

Description

```
lseek changes the position of the file read/write pointer for
the file descriptor 'fd' by 'offset' bytes.  The offset is
calculated from the value of 'origin' as follows:

SMB_SEEK_SET: seek from the beginning of file
SMB_SEEK_CUR: seek from current file position
SMB_SEEK_END: seek from end of file
```

Parameters

```
PARAMETER1: file descriptor
PARAMETER2: number of bytes to seek
PARAMETER3: mode of offset
```

Return Value

```
new file offset (from beginning of file) (>=0).

on error (<0):

_SMB_ERR_INVAL_FD         - invalid file descriptor
_SMB_ERR_INVAL_PARAMETERS - bad origin
```

Name

smb_mkdir

Synopsis

```
int smb_mkdir(int sid, char *path);
```

Description

```
Creates directory 'path' on the connection identified by sid.
```

Parameters

```
PARAMETER1: SMB Session descriptor

PARAMETER2: Full path of directory to create (./.. usage is not
            allowed).
```

Return Value

```
One of the following status codes (<0 on error):

_SMB_NOERRORS          - Search successfully closed
_SMB_ERR_INVAL_SID     - Invalid SID
_SMB_ERR_INVAL_PATH    - Invalid path
_SMB_ERR_ACCESS_DENIED - Access permissions do not allow this
                         operation
_SMB_ERR_INVAL_TID     - internal data error
_SMB_ERR_TRANSPORT     - Fatal NetBIOS error in send/recv
_SMB_ERR_TIMEOUT       - Fatal timeout waiting for response
```

See Also

```
smb_rmdir
```

Name

smb_mkutime

Synopsis

```
smb_utime_t smb_mkutime(smb_datetime *dt);
```

Description

```
Calculates UNIX time from date/time info fields in 'dt'.

UNIX time is based on the number of seconds since
Jan 1, 1970 00:00:00.0 UTC

smb_mkutime takes into account leap year (every four years
starting 2/29/1972).

typedef struct {
        uint16 sec;
        uint16 min;
        uint16 hour;    // 24 hour format
        uint16 day;     // day of month
        uint16 month;   // 1=Jan, 2=Feb, etc
        uint16 year;    // four digit digit format, ie 2003
} smb_datetime;
```

Parameters

```
PARAMETER1: pointer to smb_datetime struct to save date/time info into
```

Return Value

```
time in UNIX time format
```

See Also

smb_convutime, smb_fstat, smb_stat

Name

smb_open

Synopsis

```
int smb_open(int sid, char *name, int flags);
```

Description

```
Open a file/named resource 'name' over the SMB session on 'sid'.
Interface is similar to the C system call 'open'.  Returns a
file descriptor (integer) to be used in other SMB file I/O
commands such as smb_write, smb_read, and smb_close.
```

Parameters

```
PARAMETER1: smb session id
PARAMETER2: filename to open
PARAMETER3: flags for opening file.  flags are made up of a primary file
            mode which can be bitwise OR'd with a number of options.

            Possible file modes:

            SMB_O_RDONLY    - opened with read only privledges
            SMB_O_WRONLY    - opened with write only privledges
            SMB_O_RDWR      - opened with read/write privledges

            which can be BITWISE OR'd with any number of the following:

            SMB_O_CREAT     - if file doesn't exist, create new file
            SMB_O_EXCL      - when used with SMB_O_CREAT, open fails if file
                              exists
            SMB_O_TRUNC     - if file exists, truncate filesize to zero
                              (used with SMB_O_WRONLY or SMB_O_RDWR)
            SMB_O_APPEND    - append from EOF (used with SMB_O_WRONLY or
                              SMB_O_RDWR)
```

Return Value

```
on success (>=0):

file descriptor to be used in calls to smb_read, smb_write, and
smb_close.

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - filename null or too long
_SMB_ERR_NO_RESOURCES     - out of file descriptors
_SMB_ERR_INVAL_PATH       - invalid filename path
```

```
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

`smb_write`, `smb_read`, `smb_close`

Name

smb_perror

Synopsis

```
void smb_perror(int errcode);
```

Description

```
Prints out a diagnostic string that corresponds to the SMB
error code.

SMB error codes are enumerated in the file "smb.h"

SMB error codes are as follows:

#define _SMB_NOERRORS                 0
#define _SMB_ERR_NEGREQ              -1
#define _SMB_ERR_NEGRSVP             -2
#define _SMB_ERR_UNSUPPORTED_CMD     -3
#define _SMB_ERR_UNSUPPORTED_FLAG    -4
#define _SMB_ERR_UNSUPPORTED_DIALECT -5
#define _SMB_ERR_NO_NODE_NAME        -6
#define _SMB_ERR_NO_WRKGRP_NAME      -7
#define _SMB_ERR_TRANSPORT           -8
#define _SMB_ERR_DEST_UNREACHABLE    -9
#define _SMB_ERR_REMOTE              -10
#define _SMB_ERR_CALL                -11
#define _SMB_ERR_INVAL_SID           -12
#define _SMB_ERR_INVAL_PARAMETERS    -13
#define _SMB_ERR_INVAL_TID           -14
#define _SMB_ERR_INVAL_UID           -15
#define _SMB_ERR_INVAL_FID           -16
#define _SMB_ERR_INVAL_FD            -17
#define _SMB_ERR_INVAL_SD            -18
#define _SMB_ERR_INVAL_PATH          -19
#define _SMB_ERR_INVAL_SHARE         -20
#define _SMB_ERR_NAME_ASSIGNMENT     -21
#define _SMB_ERR_ACCESS_DENIED       -22
#define _SMB_ERR_NO_RESOURCES        -23
#define _SMB_ERR_LOCAL_BUF           -24
#define _SMB_ERR_WORDCOUNT           -25
#define _SMB_ERR_MALFORMED           -26
#define _SMB_ERR_FILE_EXISTS         -27
#define _SMB_ERR_FILE_NOT_FOUND      -28
#define _SMB_ERR_SOCKREAD            -29
#define _SMB_ERR_SOCKWRITE           -30
#define _SMB_ERR_OUT_OF_SID          -31
#define _SMB_ERR_PARAMCOUNT          -32
#define _SMB_ERR_BU_TIMEOUT          -33
#define _SMB_ERR_TIMEOUT             -34
```

```
#define _SMB_ERR_NO_MSBROWSE          -35
#define _SMB_ERR_HOST_NOT_FOUND       -36
#define _SMB_ERR_SERVICE_NOT_RUNNING  -37
#define _SMB_ERR_UNKNOWN              -38
```

Parameters

PARAMETER1: SMB error code (must be <= 0)

Return Value

none.

See Also

smb_strerr

Name

smb_ptime

Synopsis

```
void smb_ptime(smb_datetime *dt);
```

Description

```
Prints date/time info from a given smb_datetime structure.
Time will be displayed as follows:

Time format: hour:min:sec
Date format: month/day/year

ie: 02:34:17 06/21/2000

typedef struct {
        uint16 sec;
        uint16 min;
        uint16 hour;    // 24 hour format
        uint16 day;     // day of month
        uint16 month;   // 1=Jan, 2=Feb, etc
        uint16 year;    // four digit digit format, ie 2003
} smb_datetime;
```

Parameters

```
PARAMETER1: pointer to smb_datetime struct
```

Return Value

```
none.
```

See Also

```
smb_mkutime, smb_fstat, smb_stat
```

Name

smb_read

Synopsis

```
int smb_read(int fd, char *buf, int len);
```

Description

```
Read 'len' bytes from file on file descriptor 'fd' into buffer
'buf' over active SMB connection associated with 'fd'.
```

Parameters

```
PARAMETER1: file descriptor
PARAMETER2: buffer to write file data
PARAMETER3: number of bytes to to read
```

Return Value

```
Number of bytes read on success (>=0).

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - null buffer parameter
_SMB_ERR_INVAL_FD         - invalid file descriptor
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_FID        - internal data error
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

smb_write, smb_open, smb_close

Name

smb_rename

Synopsis

```
int smb_rename(int sid, char *oldname, char *newname);
```

Description

```
Renames a file over an SMB session on 'sid'.  Renames file from
'oldname' to 'newname'.
```

Parameters

```
PARAMETER1: SMB session id
PARAMETER2: file to rename
PARAMETER3: new filename
```

Return Value

```
0 on success.

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - filename(s) null or too long
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

Name

smb_resolve

Synopsis

```
uint32 smb_resolve(char *name);
```

Description

```
Looks up the IP for the given NetBIOS name.  Uses cached value,
if available.
```

Parameters

```
PARAMETER1: NetBIOS name to lookup IP for. (max length 15, truncated if
            longer)
```

Return Value

```
IP address of resolved name, or zero if resolve was
unsuccessful.
```

See Also

```
smb_resolve_force
```

Name

smb_resolve_force

Synopsis

```
uint32 smb_resolve_force(char *name);
```

Description

```
Looks up the IP for the given NetBIOS name.  Forces network
lookup (will not use cached value).
```

Parameters

```
PARAMETER1: NetBIOS name to lookup IP for. (max length 15, truncated if
            longer)
```

Return Value

```
IP address of resolved name, or zero if resolve was
unsuccessful.
```

See Also

```
smb_resolve
```

Name

smb_resolve_ip

Synopsis

```
int smb_resolve_ip(uint32 ip, char *name);
```

Description

Looks up the primary NetBIOS name for the machine at a given IP.

Parameters

```
PARAMETER1: IP to lookup NetBIOS name for.

PARAMETER2: Return value.  The primary NetBIOS machine name is written
            to 'name' on success.  Make sure 'name' is large enough to
            contain up to 16 (_NB_NAMELEN+1) characters.
```

Return Value

```
Return Codes (<0 on error):

_SMB_NOERRORS      - Name successfully resolved from IP
_SMB_ERR_TRANSPORT - No response/no names returned
```

Name

smb_rmdir

Synopsis

```
int smb_rmdir(int sid, char *path);
```

Description

```
Deletes directory 'path' on the connection identified by sid.
Target directory must be empty of files before deleting.
```

Parameters

```
PARAMETER1: SMB Session descriptor

PARAMETER2: Full path of directory to create ('.' or '..' usage is not
            allowed).
```

Return Value

```
One of the following status codes (<0 on error):

_SMB_NOERRORS          - Search successfully closed
_SMB_ERR_INVAL_SID     - Invalid SID
_SMB_ERR_INVAL_PATH    - Invalid path
_SMB_ERR_ACCESS_DENIED - Access permissions do not allow this
                         operation, or directory is not empty
_SMB_ERR_REMOTE        - Possibly tried to remove an illegal
                         directory
_SMB_ERR_INVAL_TID     - internal data error
_SMB_ERR_TRANSPORT     - Fatal NetBIOS error in send/recv
_SMB_ERR_TIMEOUT       - Fatal timeout waiting for response
```

See Also

```
smb_mkdir
```

Name

smb_sclose

Synopsis

```
int smb_sclose(smb_search *search);
```

Description

```
Closes a directory search started by smb_sopen. The buffer
supplied to smb_sopen is released back to the user.
```

Parameters

```
PARAMETER1: Pointer to local search data structure
```

Return Value

```
One of the following status codes (<0 on error):

_SMB_NOERRORS     - Search successfully closed
_SMB_ERR_INVAL_SD - Invalid search structure
```

See Also

smb_sopen, smb_sread, smb_dir_info

Name

smb_server_info

Synopsis

```
typedef struct {
        int  count;  //number of server strings in name array
        int  total;  //total number of servers available (will be
                     //   more than count if there are more servers
                     //   than will fit in user buffer OR if more
                     //   than will fit in a single smb (due to SMB
                     //   bufsize)
        char **name; //list of server names (char *name[count])
} smb_server_info;
```

Description

Data structure for use with smb_list_servers functions. Pass the address of a smb_server_info structure to one of these functions, and all fields will be filled with proper values. Count is the number of names contained in the 'name' field. Total is the total number of names available. Usually count and total will be the same, however, it can be larger for any of the following reasons:

1. there are more computers in the workgroup than will fit in a single smb (try increasing _SMB_BUFSIZ).

2. there are more computers returned from the server than will fit in the available buffer passed in (try increasing the buffer size).

See Also

smb_list_servers, smb_list_servers_ex, smb_list_servers_manual

Name

smb_set_comment

Synopsis

```
void smb_set_comment(char *comment);
```

Description

Sets the browse comment. The browse comment can be viewed by other SMB clients from Network Neighborhood or similar. It can be a maximum of SMB_BROWSE_COMMENT_SIZE and if it is longer it will be truncated.

The browse comment can either be set at compile time by defining SMB_BROWSE_COMMENT to the comment, ie:

```
#define SMB_BROWSE_COMMENT "Embeo SMB Client"
```

or it can be set via the function smb_set_comment at runtime by defining SMB_DYNAMIC_BROWSE_COMMENT_ENABLE, ie:

```
#define SMB_DYNAMIC_BROWSE_COMMENT_ENABLE
```

SMB_DYNAMIC_BROWSE_COMMENT_ENABLE and
SMB_BROWSE_ENABLE must be defined to use this function.

Parameters

PARAMETER1: Browse comment.

See Also

smb_get_comment

Name

smb_set_nodename

Synopsis

```
int smb_set_nodename(char *name);
```

Description

```
Sets the SMB client's node name to 'name'. If name is longer
than 15 characters, it is truncated. If registration is
successful, the registered name can be retrieved by calling
smb_get_nodename().
```

Return Value

```
_SMB_NOERRORS             - Node name changed successfully.
_SMB_ERR_NAME_ASSIGNMENT  - Name could not be registered.
_SMB_ERR_INVAL_PARAMETERS - Invalid/null name
```

See Also

```
smb_get_nodename
```

Name

smb_set_nodetype

Synopsis

```
int16 smb_set_nodetype(uint8 nodetype);
```

Description

```
Sets the NetBIOS node type.  In order to use this function,
the NetBIOS library must be configured for all node types by
defining NETBIOS_ALL_NODE:

#define NETBIOS_ALL_NODE

Also, in order to operate in M or P nodes a WINS IP must be set
with the function smb_set_WINS.
```

Parameters

```
PARAMETER1: Node type is one of the following:

            SMB_BNODE = Broadcast node
            SMB_MNODE = Mixed-mode node
            SMB_PNODE = Point-to-point node
```

Return Value

```
_SMB_NOERRORS             - node type changed successfully
_SMB_ERR_INVAL_PARAMETERS - bad nodetype
_SMB_ERR_ACCESS_DENIED    - bad config (define NETBIOS_ALL_NODE)
_SMB_ERR_UNKNOWN          - unknown NetBIOS error
```

See Also

smb_get_nodetype, smb_set_WINS

Name

smb_set_waitcallback

Synopsis

```
void smb_set_waitcallback(callbackfuncptr, void *userdata);
```

Description

```
Sets a wait callback function to be called when the SMB library
is waiting for a response from the server.  This function can
be useful to do work and maintain system responsiveness while
a function is blocking.

The format of the callback function is as follows:

int callbackfunction(int sid, uint32 starttime, void *userdata);

Call this function as follows:

smb_set_waitcallback(&callbackfunction, NULL);

The callback is passed the sid associated with the connection,
the start time of MS_TIMER, and a user-supplied data pointer.
If the callback returns a negative value, the SMB connection
will timeout, otherwise the system will continue waiting until
_SMB_XFER_TIMEOUT time (ms) has passed.

To handle timeouts entirely via this callback function, define
_SMB_TIMEOUT_DISABLE:

#define _SMB_TIMEOUT_DISABLE
```

Return Value

```
None.
```

See Also

Name

smb_set_WINS

Synopsis

```
void smb_set_WINS(uint32 *wins, uint16 count);
```

Description

```
Initializes the WINS server table. If 'wins' is NULL and 'count'
is zero, internal server table is reset.
```

Parameters

```
PARAMETER1: An array of NBNS server IP addresses

PARAMETER2: Number of IP's in the array
```

Return Value

```
None.
```

See Also

```
smb_get_WINS
```

Name

smb_set_wrkgrpname

Synopsis

```
int smb_set_wrkgrpname(char *name);
```

Description

```
Sets the SMB client's workgroup name. If the name is longer than
15 characters, it is truncated. If registration is successful,
the registered name can be retrieved by calling
smb_get_wrkgrpname().
```

Return Value

```
_SMB_NOERRORS             - Workgroup name changed successfully.
_SMB_ERR_NAME_ASSIGNMENT  - Name could not be registered.
_SMB_ERR_INVAL_PARAMETERS - Invalid/null name
```

See Also

```
smb_get_wrkgrpname
```

Name

smb_share_info

Synopsis

```
typedef struct {
        int  count;  //number of share strings in name array
        int  total;  //total number of shares available (will be more
                     //   than count if there are more shares than
                     //   will fit in user buffer OR if more than
                     //   will fit in a single smb due to SMB
                     //   bufsize)
        char **name; //list of shares on server (char *name[count])
} smb_share_info;
```

Description

Data structure for use with smb_list_shares functions. Pass the address of a smb_share_info structure to one of these functions, and all fields will be filled with proper values. Count is the number of names contained in the 'name' field. Total is the total number of names available. Usually count and total will be the same, however, total can be larger for any of the following reasons:

1. there are more shares on the server than will fit in a single smb (try increasing _SMB_BUFSIZ).

2. there are more shares returned from the server than will fit in the available buffer passed in (try increasing the buffer size).

See Also

smb_list_shares, smb_list_shares_ex

Name

smb_sopen

Synopsis

```
int smb_sopen(int sid, smb_search *search, char *srchpattern,
              uint16 flags, void *buffer, uint16 bufsize);
```

Description

Opens a search session on remote server identified by sid. User must supply a temporary buffer to cache returned filenames.

Parameters

PARAMETER1: SMB Session Id (returned by smb_connect)

PARAMETER2: Pointer to local search data structure. If the structure is currently in use, its current search will be closed and the new search opened.

PARAMETER3: Search pattern string (may contain wildcards)

PARAMETER4: Flags specifying type of search. The search options are:

```
SMB_DIR_SRCH_DIR    - search only for directories
SMB_DIR_SRCH_FILES  - search only for normal files
                      (no hidden/sytem)
SMB_DIR_SRCH_ALL    - search for all files and diectories
                      (including hidden/system/readonly)
```

The above are provided as shortcuts. The flags is a value created by bitwise OR'ing any number of the following:

```
_SMB_FSIO_READONLY
_SMB_FSIO_HIDDEN
_SMB_FSIO_SYSTEM
_SMB_FSIO_DIRECTORY
_SMB_FSIO_ARCHIVE
```

PARAMETER5: Pointer to user supplied buffer used to cache filenames

PARAMETER6: Size of user supplied buffer in bytes

Return Value

One of the following status codes (<0 on error):

```
_SMB_NOERRORS           - Search successfully opened
```

```
_SMB_ERR_FILE_NOT_FOUND - No matching files found
_SMB_ERR_NO_RESOURCES   - Insufficient resources to complete
                          command
_SMB_ERR_INVAL_SID      - SID is invalid
_SMB_ERR_INVAL_PATH     - Invalid search path
_SMB_ERR_INVAL_SHARE    - Share is a printer device
_SMB_ERR_INVAL_TID      - Internal session data error
_SMB_ERR_ACCESS_DENIED  - Invalid access permissions
_SMB_ERR_TRANSPORT      - Fatal NetBIOS error in send/recv
_SMB_ERR_TIMEOUT        - Fatal timeout waiting for response
```

See Also

smb_connect, smb_sread, smb_sclose, smb_dir_info

Name

smb_sread

Synopsis

```
int smb_sread(smb_search *search, smb_dir_info **files);
```

Description

```
Returns results gathered by opening a search with smb_sopen.
Repeated calls may be required to get all of the filenames
matching the search. On subsequent calls, the pointers to
previously returned filenames are to be considered invalid.
A fixed maximum number of filenames are returned with each
call.
```

Parameters

```
PARAMETER1: Pointer to local search data structure

PARAMETER2: Address of a smb_dir_info pointer (actual structure need not
            be declared, it is maintained internally). Following a
            call to this function, the pointer will point to the actual
            data filled structure. The structure is as follows:

            typedef struct {
                char count;       //number of valid filename pointers
                char *filename[]; //array of pointers to filenames
            } smb_dir_info;
```

Return Value

```
Number of filenames returned or one of the following error
codes (<0 on error):

_SMB_NOERRORS           - Read search results successfully
_SMB_ERR_FILE_NOT_FOUND - Reached end of search
_SMB_ERR_INVAL_SD       - Invalid search handle (possibly
                          closed)
_SMB_ERR_NO_RESOURCES   - Insufficient resources to complete
                          command
_SMB_ERR_INVAL_SID      - Indicates search structure data
                          corruption
_SMB_ERR_INVAL_TID      - Internal session data error
_SMB_ERR_TRANSPORT      - Fatal NetBIOS error in send/recv
_SMB_ERR_TIMEOUT        - Fatal timeout waiting for response
_SMB_ERR_INVAL_PARAMETERS - Null input parameter(s)
```

See Also

`smb_sopen`, `smb_sclose`, `smb_dir_info`

Name

smb_stat

Synopsis

```
int smb_stat(int sid, char *name, struct smb_stat_t *stat);
```

Description

```
Gets information about the file named *name and saves it in the
user-supplied smb_stat_t structure.

The smb_stat_t structure contains the file's size, attributes,
and modification time.

struct smb_stat_t {
    uint32       st_size;       // total size in bytes
    uint16       st_attributes; // DOS attributes
    smb_utime_t  st_time;       // time of last data modification
};

The modification time, st_time, can be converted to useful
terms via the smb_convutime function.
```

Parameters

```
PARAMETER1: SMB session id
PARAMETER2: filename to retrieve status on
PARAMETER3: pointer to smb_stat_t structure

            st_attributes fields:

            #define _SMB_FSIO_READONLY    0x0001
            #define _SMB_FSIO_HIDDEN      0x0002
            #define _SMB_FSIO_SYSTEM      0x0004
            #define _SMB_FSIO_VOLUME      0x0008
            #define _SMB_FSIO_DIRECTORY   0x0010
            #define _SMB_FSIO_ARCHIVE     0x0020
```

Return Value

```
0 on success.

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - filename null or too long
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
```

```
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

`smb_fstat`, `smb_convutime`

Name

smb_stat_t

Synopsis

```
struct smb_stat_t {
        uint32        st_size;        // total size in bytes
        uint16        st_attributes;  // dos attributes
        smb_utime_t   st_time;        // time of last data
                                      //    modification (mtime)
};
```

Description

Data structure for use with smb_stat and smb_fstat functions.

st_attributes can be any number of the following bit flags OR'd together:

```
#define _SMB_FSIO_READONLY    0x0001
#define _SMB_FSIO_HIDDEN      0x0002
#define _SMB_FSIO_SYSTEM      0x0004
#define _SMB_FSIO_VOLUME      0x0008
#define _SMB_FSIO_DIRECTORY   0x0010
#define _SMB_FSIO_ARCHIVE     0x0020
```

st_time can be parsed into fields by using the smb_convutime function.

See Also

smb_stat, smb_fstat

Name

smb_strerr

Synopsis

```
char *smb_strerr(int errcode);
```

Description

```
Returns a string that corresponds to the SMB error code.

The string returned is only valid until the next smb_strerr or
smb_perror call.

SMB error codes are enumerated in the file "smb.h"

SMB error codes are as follows:

#define _SMB_NOERRORS                 0
#define _SMB_ERR_NEGREQ              -1
#define _SMB_ERR_NEGRSVP             -2
#define _SMB_ERR_UNSUPPORTED_CMD     -3
#define _SMB_ERR_UNSUPPORTED_FLAG    -4
#define _SMB_ERR_UNSUPPORTED_DIALECT -5
#define _SMB_ERR_NO_NODE_NAME        -6
#define _SMB_ERR_NO_WRKGRP_NAME      -7
#define _SMB_ERR_TRANSPORT           -8
#define _SMB_ERR_DEST_UNREACHABLE    -9
#define _SMB_ERR_REMOTE              -10
#define _SMB_ERR_CALL                -11
#define _SMB_ERR_INVAL_SID           -12
#define _SMB_ERR_INVAL_PARAMETERS    -13
#define _SMB_ERR_INVAL_TID           -14
#define _SMB_ERR_INVAL_UID           -15
#define _SMB_ERR_INVAL_FID           -16
#define _SMB_ERR_INVAL_FD            -17
#define _SMB_ERR_INVAL_SD            -18
#define _SMB_ERR_INVAL_PATH          -19
#define _SMB_ERR_INVAL_SHARE         -20
#define _SMB_ERR_NAME_ASSIGNMENT     -21
#define _SMB_ERR_ACCESS_DENIED       -22
#define _SMB_ERR_NO_RESOURCES        -23
#define _SMB_ERR_LOCAL_BUF           -24
#define _SMB_ERR_WORDCOUNT           -25
#define _SMB_ERR_MALFORMED           -26
#define _SMB_ERR_FILE_EXISTS         -27
#define _SMB_ERR_FILE_NOT_FOUND      -28
#define _SMB_ERR_SOCKREAD            -29
#define _SMB_ERR_SOCKWRITE           -30
#define _SMB_ERR_OUT_OF_SID          -31
#define _SMB_ERR_PARAMCOUNT          -32
```

```
#define _SMB_ERR_BU_TIMEOUT           -33
#define _SMB_ERR_TIMEOUT              -34
#define _SMB_ERR_NO_MSBROWSE          -35
#define _SMB_ERR_HOST_NOT_FOUND       -36
#define _SMB_ERR_SERVICE_NOT_RUNNING  -37
#define _SMB_ERR_UNKNOWN              -38
```

Parameters

PARAMETER1: SMB error code (must be <= 0)

Return Value

Error description string.

See Also

smb_perror

Name

smb_tick

Synopsis

```
int smb_tick(void);
```

Description

```
Maintains the overall health of the SMB subsystem.
```

Return Value

```
_SMB_NOERRORS              - System operating normally
_SMB_ERR_NAME_ASSIGNMENT   - Node/Group name invalid
_SMB_ERR_TRANSPORT         - NetBIOS Error
```

See Also

Name

smb_unlink

Synopsis

```
int smb_unlink(int sid, char *name);
```

Description

```
unlinks a file (deletes it) from the SMB server on SMB session
id 'sid'.
```

Parameters

```
PARAMETER1: SMB session id
PARAMETER2: Filename to remove
```

Return Value

```
0 on success

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - filename null or too long
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_SHARE      - file in use/server sharing error
_SMB_ERR_MALFORMED        - success but malformed smb
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

Name

smb_utime

Synopsis

```
int smb_utime(int sid, char *name, smb_utime_t newtime);
```

Description

```
Sets the file's last modification time to 'newtime',

The modification time, newtime, can be converted to useful
terms via the smb_convutime function or created with the
smb_mkutime function.
```

Parameters

```
PARAMETER1: SMB session id
PARAMETER2: filename to set time on
PARAMETER3: new time
```

Return Value

```
0 on success.

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - filename null or too long
_SMB_ERR_INVAL_PATH       - invalid filename path
_SMB_ERR_FILE_NOT_FOUND   - file doesn't exist
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_SHARE      - bad sharing mode on file/server err
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

See Also

smb_chmod, smb_fstat, smb_convutime, smb_mkutime

Name

smb_write

Synopsis

```
int smb_write(int fd, char *buf, int len);
```

Description

```
Write 'len' bytes from buffer 'buf' into open file on file
descriptor 'fd' over active SMB connection associated with 'fd'.
```

Parameters

```
PARAMETER1: file descriptor
PARAMETER2: data to write
PARAMETER3: number of bytes to write
```

Return Value

```
Number of bytes written on success (>=0).

on error (<0):

_SMB_ERR_INVAL_PARAMETERS - null buffer parameter
_SMB_ERR_INVAL_FD         - invalid file descriptor
_SMB_ERR_INVAL_FID        - server bad file id
_SMB_ERR_ACCESS_DENIED    - bad file access permissions
_SMB_ERR_INVAL_FID        - internal data error
_SMB_ERR_INVAL_TID        - internal data error
_SMB_ERR_INVAL_UID        - internal data error
_SMB_ERR_MALFORMED        - server success but malformed smb
_SMB_ERR_TIMEOUT          - fatal timeout waiting for response
_SMB_ERR_TRANSPORT        - fatal NetBIOS error in send/recv
```

Index