

FWALL-32 Library

Firewall

**Dynamic Link Library
for Microsoft® Windows™**

Copyright © 1995 - 2003 by Distinct Corporation

All rights reserved

Table of Contents

Table of Contents.....	iii
1 Overview	5
1.1 Introduction	5
1.2 Function Summary	6
2 Reference	7
2.1 fw_abort ().....	7
2.2 fw_async_connect ()	8
2.3 fw_close ().....	11
2.4 fw_connect ().....	12
2.5 fw_get_info ().....	14
2.6 fw_get_skt ()	16
2.7 fw_listen ()	17
2.8 fw_open ()	21
2.9 fw_select ().....	22

1.2 Function Summary

fw_abort

Aborts a connection

fw_async_connect

Connects to the remote host via the firewall server asynchronously

fw_close

Closes a firewall connection

fw_connect

Connects to the remote host via the firewall

fw_get_info

Gets the connection information

fw_get_skt

Gets the socket descriptor

fw_listen

Establishes a secondary connection with the server

fw_open

Opens a firewall connection

fw_select

Determines the status of a socket

2 Reference

2.1 fw_abort ()

Description Aborts a connection.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_abort (hfwall)
```

```
    GLOBALHANDLE hfwall;           Handle of the fwall structure from fw_open
```

Remarks The function **fw_abort** aborts the connection to the remote host via the firewall server. The parameter *hfwall* must be a valid GLOBALHANDLE obtained from a call to **fw_open**. Once the connection has been successfully aborted the connection handle *hfwall* will be no more valid and the application cannot call any other firewall library functions unless it makes a new connection with a call to the function **fw_open**.

Return Value

If the function succeeds it will return FW_ABORTED and in case of failure it will return either FW_BAD_GLOBALHANDLE or an error code returned by WSAGetLastError() function.

2.2 fw_async_connect ()

Description Asynchronously connects to the remote host via the firewall server.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_async_connect (hfwall, f_param)
```

```
GLOBALHANDLE hfwall;
```

Handle of the firewall structure from
fw_open

```
fw_param far *f_param;
```

A pointer to the **fw_param** structure with all
the required parameters.

Remarks

The function **fw_async_connect** connects to the remote host via the firewall server in an asynchronous manner. After a connection is made successfully the application can read and write to the socket and the firewall server simply acts as a bridge between the local and external hosts.

The parameter *hfwall* is the handle to the firewall structure received from **fw_open**. The *f_param* is a pointer to the **fw_param** structure which will contain all the connection specific fields. The **fw_param** structure has the following fields.

```
typedef struct fw_parameters
```

```
{
```

```
char far * methods;
```

different types of methods.

```
unsigned char n_methods;
```

the number of methods

```
unsigned char add_type;
```

address type of the destination address

```
char * domain_name;
```

the domain name

```
char far * fw_host;
```

IP address of the firewall server

```
char far * dest_host;
```

IP address of the destination host

```
unsigned short fw_port
```

firewall port

```
unsigned short dest_port
```

destination port

```
char far *username;
```

user id

```
char far *passwd;
```

password

```
int socks_ver;
```

the socks version

```
char reserved [256 - 6 * sizeof (char far *) - 2 * sizeof (unsigned char) - 2 * sizeof (unsigned short) - sizeof (int)];
```

```
} fw_param;
```

Distinct firewall library supports both SOCKS version 4 and version 5 firewall server. The *socks_ver* field of the **fw_param** structure should specify the SOCKS version. The *socks_ver* field should be any or a combination of the following values:

FW_VERSION5	The Socks version is 5
FW_VERSION4	The Socks version4

If the *socks_ver* field contains both FW_VERSION5 and FW_VERSION4 then the Distinct firewall library will try to automatically determine the firewall server version and send appropriate information to the firewall server to establish a connection.

The *methods* field can point to string "0", "1", or "2" or any combination of "0", "1", "2", for example "01" or "12". Method 0 means no authentication is required, 1 means GSSAPI and 2 means a *username* and *passwd* is required. *n_methods* is the number of methods that appear in the *method* field. Note that only methods 0 and 2 are supported currently and if method 2 is specified

the application must specify a *username* and *passwd* for authentication. Note that the *methods*, *n_methods*, and *passwd* fields are required only if the application is trying to connect to a version 5 firewall server.

The field *addr_type* specifies the type of address of the remote host. The *addr_type* can be any one of the following types.

Type	Meaning
FW_ADDR_IP4	address is a version 4 IP address
FW_ADDR_DNS	address is a DNS style domain name
FW_ADDR_IP6	address is a version 6 IP address

If the *addr_type* field is of type FW_ADDR_DNS then the *domain_name* field must point to a DNS-style domain name. If the value of the *addr_type* field is FW_ADDR_IP4 or FW_ADDR_IP6 then the *dest_host* field must contain the IP address of the destination host. Note that if the SOCKS version is 4 then the *addr_type* field should always be FW_ADDR_IP4 or FW_ADDR_DNS.

The *fw_host* field should contain the address of the firewall server. The port of the firewall server must be specified in the *fw_port* field, if this field is 0 then the firewall library will use the default port 1080. The field *dest_port* must contain port number of the destination host. Both *fw_port* and the *dest_port* must be in host byte order.

The *username* should contain a valid user id if the application is trying to connect to a SOCKS version 4 firewall server or if the application has specified method 2 for authentication in case of SOCKS version 5.

Note that any string pointer of the *fw_param* structure that is not used should be set to NULL.

Once a connection has been established a message will be posted to the application with the message id same as that passed to **fw_open**. The *wParam* will be set to the GLOBALHANDLE associated with this particular firewall connection structure and the low word of *lParam* will contain a FD_CONNECT and the high word of *lParam* will contain any error code. At this point the application can call the **fw_get_info** function to get the port number and the IP address that the server assigned to connect to the target host. Check the **fw_get_info** function for more detail.

If there is any error then the high word of *lParam* will contain either any of the error codes defined by winsock or any one of the following error codes:

FW_ASYNC_SEND_FAILED	Failed to send data.
FW_ASYNC_NO_METHOD_SELECTED	None of the methods listed by the application are acceptable
FW_ASYNC_RECV_ERROR	Error in receiving data
FW_ASYNC_CONNECTION_REFUSED	Firewall server refused connection
FW_ASYNC_INVALID_ADDR	Invalid address is specified
FW_ASYNC_BAD_GLOBALHANDLE	Invalid connection handle was passed
FW_ASYNC_UNKNOWN_VERSION	Socks server version is not supported
FW_ASYNC_INVALID_USERNAME	Invalid username specified when trying to connect to SOCKS version 4
FW_ASYNC_METHOD_NOTSUPPORTED	The method specified by the application is not supported

FW_ASYNC_TIMED_OUT	Timed out while waiting for response from the firewall server
FW_ASYNC_HOSTNAME_UNRESOLVED	Failed to resolve either the fw_host or dest_host

After a successful connection whenever a network event occurs the application's window (hWnd passed to **fw_open**) will receive a message wParam (passed to **fw_open**). The wParam will contain the firewall connection handle, the application at this point can call the **fw_get_info** function to get any necessary information, see the **fw_get_info** function for more detail. The low word of lParam will contain the network event code and the high word of lParam will contain any error code. The error code can be any error code defined by winsock.

Return Value If the function succeeds it will return FW_SUCCESS. In case of any error it will return any one of the following error values.

FW_ASYNC_BAD_GLOBALHANDLE
Invalid connection handle was passed

2.3 fw_close ()

Description

Closes a firewall connection.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_close (hfwall)
```

```
GLOBALHANDLE hfwall;      The firewall structure handle from fw_open
```

Remarks

The **fw_close** function closes a firewall connection and frees any space associated with this connection.

Return Value The **fw_close** function returns TRUE, if successful. If the firewall structure handle specified is invalid it will return FW_BAD_GLOBALHANDLE.

2.4 fw_connect ()

Description Connects to the remote host via the firewall server.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_connect (hfwall, f_param)
```

GLOBALHANDLE <i>hfwall</i> ;	Handle of the firewall structure from fw_open
fw_param far * <i>f_param</i> ;	A pointer to the fw_param structure with all the required parameters.

Remarks The function **fw_connect** connects to the remote host via the firewall server synchronously. After a connection is made successfully the application can read and write to the socket and the firewall server simply acts as a bridge between the local and external hosts.

The parameter *hfwall* is the handle to the firewall structure received from **fw_open**. The *f_param* is a pointer to the **fw_param** structure which will contain all the connection specific fields. The **fw_param** structure has the following fields.

```
typedef struct fw_parameters
{
    char far * methods;                different types of methods.
    unsigned char n_methods;           the number of methods
    unsigned char add_type;            address type of the destination address
    char * domain_name;               the domain name
    char far * fw_host;                IP address of the firewall server
    char far * dest_host;              IP address of the destination host
    unsigned short fw_port;            firewall port
    unsigned short dest_port;          destination port
    char far * username;               valid user id
    char far * passwd;                 user password
    int socks_ver;                     the socks version
    char reserved [256 - 4 * sizeof (char far *) - 2 * sizeof (unsigned char) - 2 * sizeof (unsigned short) - sizeof (int)];
} fw_param;
```

Distinct firewall library supports both SOCKS version 4 and version 5. The *socks_ver* field of the **fw_param** structure should specify the SOCKS version. The *socks_ver* field should be any or a combination of the following values:

FW_VERSION5	The Socks version is 5
FW_VERSION4	The Socks version4

If the *socks_ver* field contains both FW_VERSION5 and FW_VERSION4 then the Distinct firewall library will try to automatically determine the firewall server version and send appropriate information to the server to establish a connection.

The *methods* field can point to string "0", "1", or "2" or any combination of "0", "1", "2", for example "01" or "12". Method 0 means no authentication is required, 1 means GSSAPI and 2 means a *username* and *passwd* is required. *n_methods* is the number of methods that appear in the *method* field. Note that only methods 0 and 2 are supported currently and if method 2 is specified the application must specify a *username* and *passwd* for authentication. Note that the *methods*, *n_methods*, and *passwd* fields are required only if the application is trying to connect to a version 5 firewall server.

The field *addr_type* specifies the type of address in the *dest_host* field. The *addr_type* can be any one of the following types.

Type	Meaning
FW_ADDR_IP4	address is a version 4 IP address
FW_ADDR_DNS	address is a DNS style domain name
FW_ADDR_IP6	address is a version 6 IP address

If the *addr_type* field is of type FW_ADDR_DNS then the *domain_name* field must point to a DNS-style domain name. If the value of the *addr_type* field is FW_ADDR_IP4 or FW_ADDR_IP6 then the *dest_host* field must contain the IP address of the destination host. Note that if the SOCKS version is 4 then the *addr_type* field should be either FW_ADDR_IP4 or FW_ADDR_DNS.

Note that any string pointer of the *fw_param* structure that is not used should be set to NULL.

The *fw_host* field should contain the address of the firewall server. The port of the firewall server must be specified in the *fw_port* field, if this field is 0 then the firewall library will use the default port 1080. The field *dest_port* must contain port number of the destination host. Both *fw_port* and the *dest_port* must be in host byte order.

The *username* should contain a valid user id if the application is trying to connect to a SOCKS version 4 firewall server or if the application has specified method 2 for authentication in case of SOCKS version 5.

After the function returns the application can pass the firewall connection handle (*hfwall*) to the function **fw_get_info** to know the port number and the IP address that the server assigned to connect to the target host.

Return Value If the function succeeds it will return 0. In case of any error it will return any one of the following error values.

FW_SEND_FAILED	Failed to send data.
FW_NO_METHOD_SELECTED	None of the methods listed by the application are acceptable
FW_RECV_ERROR	Error in receiving data
FW_CONNECTION_REFUSED	Firewall server refused connection
FW_INVALID_ADDR	Invalid address is specified
FW_BAD_GLOBALHANDLE	Invalid connection handle was passed
FW_HOSTNAME_UNRESOLVED	Failed to resolve either the <i>fw_host</i> or <i>dest_host</i>
FW_INVALID_USERNAME	Invalid username has been specified when trying to connect to SOCK version 4
FW_UNKNOWN_VERSION	SOCKS version is unknown
FW_TIMED_OUT	Timed out while waiting for response from the firewall server.

2.5 fw_get_info ()

Description Gets a firewall connection information.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_get_info (hfwall, field, buf, size)
```

GLOBALHANDLE hfwall;	The firewall structure handle from fw_open
int field;	The field that needs to be fetched
void *buf;	Buffer where the information will be copied
int * size;	The size of the buffer

Remarks The **fw_get_info** function gets the value of a particular field from the firewall connection structure. Whenever a message is posted to the application's window with the wParam set to the firewall connection handle the application can use this function to know the port number and associated IP address assigned by the firewall server to connect to the target host, the port number assigned by the firewall server for listening or the port number and the IP address of the connecting host.

The *hfwall* is the connection specific handle that was returned by **fw_open**.

The *field* parameter should specify the field that the application wants retrieve and it must be any one of the following values:

FW_ADDR_TYPE	The address type
FW_ADDR	The address
FW_DOMAIN_NAME	The domain name
FW_SOCKET	The socket descriptor
FW_PORT	The port number

FW_ADDR_TYPE will fetch the address type, it can be any one of the following values:

Type	Meaning
FW_ADDR_IP4	address is a version 4 IP address
FW_ADDR_DNS	address is a DNS style domain name
FW_ADDR_IP6	address is a version 6 IP address

The address type will be returned in the *buf* parameter, the *buf* should be a pointer to an integer and the *size* should be the size of an integer.

If the FW_ADDR_TYPE is FW_ADDR_DNS, the application must try to get the domain name of the server by setting the *field* parameter to FW_DOMAIN_NAME and in this case the *buf* should be a char * pointing to an empty string and the *size* should specify the number of bytes that can be copied to *buf*.

If the FW_ADDR_TYPE is FW_ADDR_IP4 then the application can get the address of the server by setting the field parameter to FW_ADDR. The *buf* should be long pointer and the *size* should be the size of a long.

To get the socket descriptor the field should be FW_SOCKET, the *buf* should be a pointer to an integer and the *size* should be the size of an integer.

To get the port number the field should be FW_PORT, the *buf* should be a pointer to a short and the *size* should be the size of a short.

Return Value If the function succeeds it will return FW_SUCCESS. In case of any error it will return any one of the following error values.

FW_BAD_GLOBALHANDLE

Invalid connection handle was passed

FW_INVALID_FIELD

The field specified in the field parameter is not valid

FW_INSUFFICIENT_BUFSIZE

The buffer size is insufficient for the data, in this case the application should check the size parameter for the required buffer size.

2.6 fw_get_skt ()

Description Gets a socket descriptor.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_get_skt (hfwall)
```

```
GLOBALHANDLE hfwall;      The firewall structure handle from fw_open
```

Remarks The **fw_get_skt** function returns the socket descriptor associated with this particular connection structure.

Return Value The function **fw_get_skt** returns the socket descriptor.

2.7 fw_listen ()

Description Establishes a secondary connection after a primary connection has been made with **fw_connect**.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_listen (hfwall, f_param)
```

```
    GLOBALHANDLE hfwall;
```

Handle of the firewall structure from
fw_open

```
    fw_param far *f_param;
```

A pointer to the **fw_param** structure with all
the required parameters.

Remarks

The function **fw_listen** actually does a bind. The function is a non-blocking call and returns immediately issuing a BIND request. The BIND request is used in protocols which require the client to accept connections from the server. FTP is a well-known example.

The parameter *hfwall* is the handle to the firewall structure obtained from **fw_open**.

The *f_param* is a pointer to the **fw_param** structure which will contain all the connection specific fields. The **fw_param** structure has the following fields.

```
typedef struct fw_parameters
```

```
{
```

```
    char far * methods;
```

different types of methods.

```
    unsigned char n_methods;
```

the number of methods

```
    unsigned char add_type;
```

address type of the destination address

```
    char * domain_name;
```

the domain name

```
    char far * fw_host;
```

IP address of the firewall server

```
    char far * dest_host;
```

IP address of the destination host

```
    unsigned short fw_port
```

firewall port

```
    unsigned short dest_port
```

destination port

```
    char * username;
```

valid user id

```
    char * passwd;
```

a valid password

```
    char reserved [256 - 6 * sizeof (char far *) - 2 * sizeof (unsigned char) - 2 * sizeof ( unsigned short)];
```

```
} fw_param;
```

Distinct firewall library supports both SOCKS version 4 and version 5. The *socks_ver* field of the **fw_param** structure should specify the SOCKS version. The *socks_ver* field should be any or a combination of the following values:

FW_VERSION5	The Socks version is 5
FW_VERSION4	The Socks version4

If the *socks_ver* field contains both FW_VERSION5 and FW_VERSION4 then the Distinct firewall library will try to automatically determine the firewall server version and send appropriate information to the server to establish a connection.

The *methods* field can point to string "0", "1", or "2" or any combination of "0", "1", "2", for example "01" or "12". Method 0 means no authentication is required, 1 means GSSAPI and 2 means a *username* and *passwd* is required. *n_methods* is the number of methods that appear in the *method* field. Note that only methods 0 and 2 are supported currently and if method 2 is specified the application must specify a *username* and *passwd* for authentication. Note that the *methods*, *n_methods*, and *passwd* fields are required only if the application is trying to connect to a version 5 firewall server.

The field *addr_type* specifies the type of address in the *dest_host* field. The *addr_type* can be any one of the following types.

Type	Meaning
FW_ADDR_IP4	address is a version 4 IP address
FW_ADDR_DNS	address is a DNS style domain name
FW_ADDR_IP6	address is a version 6 IP address

If the *addr_type* field is of type FW_ADDR_DNS then the *domain_name* field must point to a DNS-style domain name. If the value of the *addr_type* field is FW_ADDR_IP4 or FW_ADDR_IP6 then the *dest_host* field must contain the IP address of the destination host. Note that if the SOCKS version is 4 then the *addr_type* field should be either FW_ADDR_IP4 or FW_ADDR_DNS.

The *fw_host* field should contain the address of the firewall server. The port of the firewall server must be specified in the *fw_port* field, if this field is 0 then the firewall library will use the default port 1080. The field *dest_port* must contain port number of the destination host. Both *fw_port* and the *dest_port* must be in host byte order.

The *username* should contain a valid user id if the application is trying to connect to a SOCKS version 4 firewall server or if the application has specified method 2 for authentication in case of SOCKS version 5.

Note that any string pointer of the *fw_param* structure that is not used should be set to NULL.

Two replies are sent from the firewall server during a BIND operation. The first request is sent after the client creates and binds to the new socket. As soon as the first reply arrives a message is posted to the application window with the same message id that the application had passed to **fw_open**. The *wParam* set to the GLOBALHANDLE associated with this particular firewall connection structure and the *lParam* set to 0. The firewall connection structure is as follows.

```
typedef struct _fwall
{
    char domain_name [256];           /* DNS style domain name */
    unsigned int user_upcall;        /* The upcall message id */
    unsigned short port;             /* The port number */
    unsigned long addr;              /* address of the firewall server or destination host*/
    int socket;                       /* The socket descriptor */
    int status;                       /* Connection status */
    int a_type;                       /* The address type */
    HWND handle;                      /* firewall library window handle */
    HWND app_hwnd;                   /* The application window handle */
    fw_param *fw;                    /* The firewall connection structure /
    HANDLE hHandle;                  /* Internal window handle /
    char *reply;                      /* Buffer for internal use */
    char reserved [1024 - 256 - sizeof ( unsigned int) - sizeof ( unsigned short) - sizeof (unsigned
long) - 3 * sizeof (int) - 2 * sizeof (HWND) - sizeof (HANDLE) - 8];
} fwall;
```

The only fields useful to the application at this point are the *domain_name*, *port*, *addr* and *a_type*. Depending on the address type returned by the server the address of the firewall server will be either in the *domain_name* field or in the *addr* field. If the *a_type* field contains FW_ADDR_IP4 then the address will be in the *addr* field, if it is FW_ADDR_DNS then the *domain_name* field will have the domain name of the server. The port field will contain the port number that the firewall server assigned to listen for an incoming connection. The application should use these pieces of information to notify the other side of the rendezvous address. Note that the application can use the new firewall library API **fw_get_info** to get all the above described informations, check the **fw_get_info** function for more detail. If everything goes well another message will be posted to the

application with the wParam and lParam both set to 0. In case of any error in receiving the first reply a message will be posted to the application with the wParam set to 0 and the lParam set to an error number (the errors are described below).

The second reply arrives only when the anticipated incoming connection succeeds or fails. On the arrival of the second reply a message will be again posted to the application window with the wParam set to the GLOBALHANDLE of the firewall connection structure and the lParam set to 0L. The fwall struct will contain the address and port number of the connecting host. If there is any error then a message will be posted with the wParam set to 0 and lParam to an error number.

After the second reply arrives no more messages will be posted to the application's window, the connection then becomes synchronous.

Return Value The function **fw_listen** always returns TRUE. Messages are posted later on with the lParam set to 0L incase of success or to an error number incase of failure. The error values are as follows.

FW_SEND_FAILED
Failed to send data.

FW_NO_METHOD_SELECTED
None of the methods listed by the application are acceptable

FW_RECV_ERROR
Error in receiving data

FW_CONNECTION_REFUSED
Firewall server refused connection

FW_INVALID_ADDR
Invalid address is specified

FW_BAD_GLOBALHANDLE
Invalid connection handle was passed

FW_HOSTNAME_UNRESOLVED
Failed to resolve either the fw_host or dest_host name

FW_TIMED_OUT
Timed out while waiting for response from the firewall server.

FW_INVALID_USERNAME

Invalid username has been specified when trying to connect to SOCK version 4

FW_UNKNOWN_VERSION

SOCKS version is unknown

2.8 fw_open ()

Description Creates a socket.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
GLOBALHANDLE FAR PASCAL fw_open (hWin, upcall)
```

HWND <i>hWin</i>	The application window handle
unsigned int <i>upcall</i> ;	Message Id that the window will receive.

Remarks The function fw_open creates a socket. The parameter *hWin* must a valid application window handle that will receive all the messages posted by the firewall library. The *upcall* is the message id that the window will receive when they call the function **fw_async_connect** or **fw_listen**.

Return Value The function fw_open returns a GLOBALHANDLE to the struct fwall which is unique for every connection. This GLOBALHANDLE must be used in all the subsequent calls. If there is any error the function will return NULL.

2.9 fw_select ()

Description Returns the status of a socket.

```
#include <windows.h>
```

```
#include <d32-fw.h>
```

```
int far pascal fw_select (hfwall)
```

```
    GLOBALHANDLE hfwall;           Handle of the fwall structure from fw_open
```

Remarks The function fw_select returns the current status of a socket. The parameter *hfwall* must be a valid GLOBALHANDLE obtained from a call to **fw_open**.

Return Value

If the function succeeds it will return any one of the following values.

FW_CONNECTED

The socket is connected.

FW_PENDING_REPLY

Waiting for reply

FW_NOTCONNECTED

The socket is not connected

If the fwall structure handle passed is invalid the function will return FW_BAD_GLOBALHANDLE.