

POP-32 Library

**Post Office Protocol
Dynamic Link Library
for Microsoft® Windows™**

Version 5.2

**Copyright © 1994 - 2003 by Distinct Corporation
All rights reserved**

Table of Contents

Table of Contents.....	iii
1 Overview	5
1.1 Introduction	5
1.2 Registry Entries	5
1.3 Function Summary	6
2 Reference	7
2.1 pop_dele ().....	7
2.2 pop_direct ().....	8
2.3 pop_fw_open ().....	9
2.4 pop_list ().....	11
2.5 pop_noop ().....	13
2.6 pop_open ().....	14
2.7 pop_opts ().....	16
2.8 pop_quit ().....	18
2.9 pop_retr ().....	19
2.10 pop_rply ().....	21
2.11 pop_rset ().....	22
2.12 pop_stat ().....	23
2.13 pop_uniq ().....	24
2.14 pop_xopen ().....	26
2.15 pop_xquit ().....	28

1 Overview

1.1 Introduction

The Distinct POP-32 (Post Office Protocol) library provides developers with a consistent API to both version 2 and version 3 of POP. POP is a service which usually runs on mail servers (such as UNIX workstations) to allow mail clients (usually PCs) to retrieve mail stored for them in their private mail box.

Once a connection has been established with a server using the `pop_(x)open` call, information about the available messages in this mail box can be obtained with the `pop_stat` and `pop_list` functions. The `pop_stat` function determines the number of messages and their total size in bytes and the `pop_list` function retrieves the size in bytes of each available message. Messages can be read with the `pop_retr` function and, once they are downloaded to the PC, removed from the server mail box with the `pop_dele` function.

When a connection is no longer needed, it must be closed with a call to `pop_quit`. Connections should not be kept open when not in use since they consume valuable system resources. An application must close all open connections before it terminates.

Most functions in the POP-32 library wait for a reply from the POP server before returning to the caller. Because communication links or servers sometimes go down, these functions use a default timeout of 20 seconds before returning the error value `POP_TIMED_OUT` if no response was received. This timeout value can be changed by editing or creating a registry entry as described in the section '1.2 - Registry Entries'.

While waiting for a reply to a command or while waiting for a data transfer to complete, most functions in the POP-32 library yield control to Windows. This allows applications to process messages while a POP operation is in progress. Without yielding, the system would appear to be frozen for the entire duration of each POP operation. By default, yielding is therefore enabled but can be disabled if necessary with the `pop_opts` call. Yielding control to Windows is done by the POP-32 library by frequently calling an internal yielding function during lengthy operations. The calling application can also define its own yielding function to be called instead of the default yielding function.

1.2 Registry Entries

Various parameters used to control the behavior of the Distinct POP-32 library are stored under the following registry path.

HKEY_LOCAL_MACHINE\SOFTWARE\Distinct\DLLS\POP32

The following entries specify various Distinct POP-32 library settings.

Timeout **REG_DWORD** *1-999*

Specifies the default timeout in seconds used while connecting to the remote POP server and for other POP operations. The default value is 20 seconds.

DebugLevel **REG_DWORD** *0-2*

Specifies what debug information should be logged into the D32-POP.DBG file in the directory from which D32-POP.DLL was loaded. The following values are supported.

Value	Meaning
0	Disable logging.
1	Log all errors.
2	Log all function calls and return values.

A higher debug level automatically includes all lower debug levels. The default value is 0.

1.3 Function Summary

pop_dele

Delete specified message

pop_direct

Send a command to the POP server

pop_fw_open

Open connection with POP server via the firewall

pop_list

Get listing of available messages

pop_noop

Do nothing

pop_open

Open connection with POP server

pop_opts

Set connection option

pop_quit

Close connection with POP server

pop_retr

Retrieve specified message

pop_rply

Get last server reply

pop_rset

Reset POP server

pop_stat

Get statistics about available messages

pop_uniq

Unique message id

pop_xopen

Asynchronous version of pop_open

pop_xquit

Cancel asynchronous connect or operation in progress

2 Reference

2.1 pop_dele ()

Description Delete specified message.

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

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

```
int WINAPI pop_dele (hConn, msg)
```

```
    GLOBALHANDLE hConn; Handle of POP structure from pop_(x)open
```

```
    int msg; Message to delete
```

Remarks The **pop_dele** function deletes the mail message *msg* from the mail box on the POP server. There is no way to undelete a message. The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Return Value The **pop_dele** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_FAILURE

Host was not able to execute command.

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data.

POP_TIMED_OUT

Time out occurred while waiting for a response from host.

POP_INVALID_MESSAGE

Specified message number was not valid (POP 2 only).

POP_IN_PROGRESS

Another operation is already in progress.

POP_ABORTED

Operation was aborted and connection has been closed.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.2 pop_direct ()

Description Send a command to the POP server.

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

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

```
int WINAPI pop_direct (hConn, msg, rep_buf, len)
```

```
GLOBALHANDLE hConn;      Handle of POP structure
```

```
LPSTR msg;               Command to send
```

```
char *rep_buf;           Buffer for command response
```

```
int *len;                Length of response buffer
```

Remarks This function sends the command specified by *msg* to the POP server. The response returned by the server is placed into the buffer pointed to by *rep_buf*. The size of the response buffer is specified by *len*. The return buffer is NULL terminated. If *rep_buf* is not large enough for the entire response then the message from the POP server will be truncated and the *len* parameter will contain the required length of the *rep_buf*.

Note: This function is supplied to allow the user to send any commands that are not supported by the POP Library. This method is not a substitute for the existing functions and should not be used as such. In particular this function should not be used to disconnect as the POP Library will not be aware of the action. This function cannot be used until a successful connection has been made with `pop_open` or `pop_fw_open`.

Return Value The `pop_direct` function either returns `POP_SUCCESS` or one of the following errors.

`POP_INVALID_HANDLE`

Invalid connection handle was specified.

`POP_FAILURE`

Host was not able to execute command.

`POP_CANNOT_SEND_COMMAND`

Windows Sockets library was not able to accept outgoing data.

`POP_TIMED_OUT`

Time out occurred while waiting for a response from host.

`POP_IN_PROGRESS`

Another operation is already in progress.

`POP_ABORTED`

Operation was aborted and connection has been closed.

`POP_INSUFFICIENT_LENGTH`

The reply buffer (*rep_buf*) was not large enough to hold the entire server response, the *len* parameter will contain the required length of the buffer.

If this function returns `POP_FAILURE`, then the `pop_rply` function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.3 pop_fw_open ()

Description Open connection with POP server via firewall.

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

int FAR PASCAL pop_fw_open (type, usr, pwd, conn, proc, lParam, fw)

    int type;                Type of POP server
    char far *usr;           User name
    char far *pwd;           Password
    GLOBALHANDLE far *conn;  Buffer for handle of POP structure
    FARPROC proc;            Application callback function
    DWORD lParam;           Application supplied parameter
    fw_param far *fw;        A pointer to the fw_param structure
```

Remarks The `pop_fw_open` function establishes a TCP/IP connection with the specified host via the firewall. The type of POP server must be specified by setting `type` to either one of the following two values.

TYPE_POP2	POP version 2
TYPE_POP3	POP version 3

Most servers support the more advanced POP 3 protocol. Once the server type has been set, all remaining functions in the Distinct POP Library will work without specifying the version of POP.

Once a connection has been established via the firewall, the caller is automatically logged into the POP server with the user name `usr` and the password `pwd`. After a successful login, the handle of the connection control structure is placed into the buffer pointed to by `conn`. This handle must be used in all subsequent calls to the other POP functions for this connection. The Distinct POP Library supports multiple concurrent POP connections limited only by system resources.

During development it may be necessary to observe exactly what commands are sent to the POP server and what replies are generated. For this purpose, the application can specify the address of a debug callback function with `proc`. Note that `proc` must be obtained by calling **MakeProcInstance** specifying an exported function. This function will receive a copy of all network traffic and must be defined as follows.

```
int FAR PASCAL (*proc) (hConn, buf, len, lParam)

GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

The `proc` parameter must be set to NULL if the application does not want to receive debug messages.

The `lParam` parameter can be set to any value that has meaning to the application such as an index into an array or a pointer to a structure. This parameter is passed back to the application as an argument in each callback function and allows the application to pass connection specific data to the callback function.

The `fw` parameter is a pointer to the **fw_param** firewall structure which must contain all the connection specific fields required to establish connection to the POP server via the firewall. 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 SMTP server
    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 - 6 * sizeof (char far *) - 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 version is 4

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 then 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 *add_type* specifies the type of address of the remote host. The *add_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 *add_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`.

Return Value The `pop_fw_open` function either returns `POP_SUCCESS` or one of the following errors.

`POP_FAILURE`

Invalid user name or password specified.

`POP_CANNOT_INITIALIZE`

Support libraries or network drivers could not be initialized.

`POP_CANNOT_INIT_WINSOCK`

Windows Sockets transport library could not be initialized.

`POP_OUT_OF_MEMORY`

Not enough memory to allocate connection control structure.

`POP_UNKNOWN_HOST`

Unable to resolve specified host name.

`POP_CANNOT_ALLOC_SOCKET`

Windows Sockets library could not allocate a socket.

`POP_CANNOT_BIND_SOCKET`

Windows Sockets library was not able to bind socket to local port.

`POP_HOST_NOT_RESPONDING`

Unable to connect to POP port on specified host.

`POP_CANNOT_SEND_COMMAND`

Windows Sockets library was not able to accept outgoing data.

`POP_TIMED_OUT`

Time out occurred while waiting for a response from host.

If this function returns `POP_FAILURE`, then the `pop_rply` function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.4 pop_list ()

Description Get listing of available messages.

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

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

```
int WINAPI pop_list (hConn, proc)
```

```
    GLOBALHANDLE hConn; Handle of POP structure from pop_(x)open
    FARPROC proc;       Application callback function
```

Remarks The **pop_list** function provides the caller with a list of all messages available in the mail box on the POP server. For each available message, the callback function *proc* is called with a string specifying the message number and the length of the message in bytes. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, line, len, lParam)
```

```
GLOBALHANDLE hConn;
char far *line;
int len;
DWORD lParam;
```

An application can abort the listing by returning -1 in the callback function (all other return values are currently ignored) or by calling **pop_xquit**. Note that aborting a listing will also abort the underlying connection by resetting it. If the listing was aborted successfully, then the **pop_list** call will return POP_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **pop_quit**) can be made until a new connection has been established with **pop_(x)open**.

The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Note that this command will not work properly on a POP 2 connection after the **pop_dele** function has been called one or more times.

Return Value The **pop_list** function either returns POP_SUCCESS or one of the following errors.

```
POP_INVALID_HANDLE
```

Invalid connection handle was specified.

```
POP_FAILURE
```

Host was not able to execute command.

```
POP_CANNOT_SEND_COMMAND
```

Windows Sockets library was not able to accept outgoing data.

```
POP_TIMED_OUT
```

Time out occurred while waiting for a response from host.

```
POP_IN_PROGRESS
```

Another operation is already in progress.

```
POP_ABORTED
```

Operation was aborted and connection has been closed.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.5 pop_noop ()

Description Do nothing.

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

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

```
int WINAPI pop_noop (hConn)
```

GLOBALHANDLE *hConn*; Handle of POP structure from **pop_(x)open**

Remarks The **pop_noop** function sends an empty command to the POP server and waits for a proper reply. This command is only supported on a POP 3 connection where it can be used to verify that the server is active. On a POP 2 connection, this command returns immediately. The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Return Value The **pop_noop** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_FAILURE

Host was not able to execute command (POP 3 only).

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data (POP 3 only).

POP_TIMED_OUT

Time out occurred while waiting for a response from host (POP 3 only).

POP_IN_PROGRESS

Another operation is already in progress.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.6 pop_open ()

Description Open connection with POP server.

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

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

```
int WINAPI pop_open (host, type, usr, pwd, conn, proc, lParam)
```

LPSTR <i>host</i> ;	Name of POP server
int <i>type</i> ;	Type of POP server
LPSTR <i>usr</i> ;	User name
LPSTR <i>pwd</i> ;	Password
GLOBALHANDLE far * <i>conn</i> ;	Buffer for handle of POP structure
FARPROC <i>proc</i> ;	Application callback function
DWORD <i>lParam</i> ;	Application supplied parameter

Remarks The **pop_open** function establishes a TCP/IP connection with the specified host. The *host* parameter can point either to a machine name (possibly including a domain name) or to an internet address in dotted decimal notation (e.g. "192.1.2.3"). The type of POP server must be specified by setting *type* to either one of the following two values.

TYPE_POP2	POP version 2
TYPE_POP3	POP version 3

Most servers support the more advanced POP 3 protocol. Once the server type has been set, all remaining functions in the Distinct POP-32 library will work without specifying the version of POP.

Once a connection has been established, the caller is automatically logged into the POP server with the user name *usr* and the password *pwd*. After a successful login, the handle of the connection control structure is placed into the buffer pointed to by *conn*. This handle must be used in all subsequent calls to the other POP functions for this connection. The Distinct POP-32 library supports multiple concurrent POP connections limited only by system resources.

During development it may be necessary to observe exactly what commands are sent to the POP server and what replies are generated. For this purpose, the application can specify the address of a debug callback function with *proc*. This function will receive a copy of all network traffic and must be defined as follows.

```
int WINAPI (*proc) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

The *proc* parameter must be set to NULL if the application does not want to receive debug messages.

The *lParam* parameter can be set to any value that has meaning to the application such as an index into an array or a pointer to a structure. This parameter is passed back to the application as an argument in each callback function and allows the application to pass connection specific data to the callback function.

Return Value The **pop_open** function either returns POP_SUCCESS or one of the following errors.

POP_FAILURE

Invalid user name or password specified.

POP_CANNOT_INITIALIZE

Support libraries or network drivers could not be initialized.

POP_CANNOT_INIT_WINSOCK

Windows Sockets transport library could not be initialized.

POP_OUT_OF_MEMORY

Not enough memory to allocate connection control structure.

POP_UNKNOWN_HOST

Unable to resolve specified host name.

POP_CANNOT_ALLOC_SOCKET

Windows Sockets library could not allocate a socket.

POP_CANNOT_BIND_SOCKET

Windows Sockets library was not able to bind socket to local port.

POP_HOST_NOT_RESPONDING

Unable to connect to POP port on specified host.

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data.

POP_TIMED_OUT

Time out occurred while waiting for a response from host.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.7 pop_opts ()

Description Set connection option.

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

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

```
int WINAPI pop_opts (hConn, option, value)
```

GLOBALHANDLE <i>hConn</i> ;	Handle of POP structure from pop_(x)open
WORD <i>option</i> ;	Option to set
DWORD <i>value</i> ;	Value of option

Remarks The **pop_opts** function sets options on a per connection basis. The *option* parameter specifies which option to set and must be equal to one of the following constants.

OPT_YIELD_ON	Enable yielding.
OPT_YIELD_OFF	Disable yielding.
OPT_SET_PARAM	Change callback parameter.
OPT_GET_PARAM	Get current callback parameter.
OPT_DEBUG	Change debug callback.
OPT_TIMEOUT	Change timeout value.

When enabling yielding with the OPT_YIELD_ON option, the *value* parameter can be used to specify the address of an application supplied yielding callback function. This function will be called during lengthy operations and should yield control to Windows so that other applications can process their messages. The function must be defined as follows.

```
int WINAPI (*proc) (void)
```

If an application wants to use the default POP-32 library yielding function, then the *value* parameter must be set to NULL. The default yielding function should be sufficient in most cases.

The *value* parameter is not used for the OPT_YIELD_OFF option and must be set to NULL.

The callback parameter supplied by the application in the **pop_(x)open** call can be changed with the OPT_SET_PARAM option. To do so, set *value* to the new callback parameter. The current value of the callback parameter can be obtained with the OPT_GET_PARAM option. In this case, *value* must point to the DWORD which is to receive the current callback parameter (i.e. *value* must be a DWORD far *).

The OPT_DEBUG option allows the application to change or disable the debug callback function. To change the debug callback function, set *value* to the address of the new callback function. Note that this function address must be obtained by calling **MakeProcInstance** specifying an exported function. To disable debug callbacks, set *value* to NULL. Refer to the **pop_(x)open** description for more details on debug callback functions.

The OPT_TIMEOUT option allows the application to set a timeout value other than the default for an open connection. The *value* parameter must specify the new timeout value in seconds and must be in the range of 1 to 999.

The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Return Value The `pop_opts` function either returns `POP_SUCCESS` or one of the following errors.

`POP_INVALID_HANDLE`

Invalid connection handle was specified.

`POP_INVALID_OPTION`

Invalid connection option was specified.

2.8 pop_quit ()

Description Close connection with POP server.

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

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

```
int WINAPI pop_quit (hConn)
```

GLOBALHANDLE *hConn*; Handle of POP structure from **pop_(x)open**

Remarks The **pop_quit** function terminates a POP connection and frees all associated resources. The connection handle *hConn* must have been obtained with a call to **pop_(x)open** and will be invalid once the **pop_quit** function returns.

Return Value The **pop_quit** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_IN_PROGRESS

Another operation is already in progress.

2.9 pop_retr ()

Description Retrieve specified message.

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

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

```
int WINAPI pop_retr (hConn, msg, lines, KeepMsg, proc)
```

GLOBALHANDLE *hConn*; Handle of POP structure from **pop_(x)open**

int *msg*; Message number to retrieve

int *lines*; Lines of message to retrieve

BOOL *KeepMsg*; Save or delete message

FARPROC *proc*; Application callback function

Remarks The **pop_retr** function retrieves some or all of the message *msg* in the mail box on the POP server. The parameter *lines* controls how many lines of the message body should be read (the message header is always read). To read the entire message, *lines* must be set to ALL_LINES. On a POP 2 connection, this parameter is ignored because the entire message is always read. Once a message has been retrieved, it can be deleted by setting the *KeepMsg* flag to FALSE. If this flag is set to TRUE, then the message can still be deleted later by calling **pop_delete**.

The message header and the specified number of lines of message body are delivered to the application with one or more calls to the callback function *proc*. This function must be defined as follows.

```
int WINAPI (*proc) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
```

```
char far *buf;
```

```
int len;
```

```
DWORD lParam;
```

An application can abort the transfer of a message by returning -1 in the callback function (all other return values are currently ignored) or by calling **pop_xquit**. Note that aborting a message transfer will also abort the underlying connection by resetting it. If the transfer was aborted successfully, then the **pop_retr** call will return POP_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **pop_quit**) can be made until a new connection has been established with **pop_(x)open**.

Once the message has been transferred, one final call to *proc* is made with the *len* parameter set to zero to indicate the end of the message.

Return Value The **pop_retr** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_FAILURE

Host was not able to execute command.

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data.

POP_TIMED_OUT

Time out occurred while waiting for a response from host.

POP_INVALID_MESSAGE

Specified message number was not valid (POP 2 only).

POP_IN_PROGRESS

Another operation is already in progress.

POP_ABORTED

Operation was aborted and connection has been closed.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.10 pop_rply ()

Description Get last server reply.

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

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

```
int WINAPI pop_rply (hConn, reply, len)
```

```
    GLOBALHANDLE hConn; Handle of POP structure from pop_(x)open
```

```
    char far *reply; Buffer for last server reply
```

```
    int len; Size of reply buffer
```

Remarks The **pop_rply** function retrieves the last response sent by the POP server. This can be useful in determining the cause of a POP_FAILURE error return. The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Return Value The **pop_rply** function either returns POP_SUCCESS or POP_INVALID_HANDLE to indicate that an invalid connection handle was specified.

2.11 pop_rset ()

Description Reset POP server.

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

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

```
int WINAPI pop_rset (hConn)
```

GLOBALHANDLE *hConn*; Handle of POP structure from **pop_(x)open**

Remarks The **pop_rset** function resets the POP server to its original state. This command is not supported on a POP 2 connection where it returns immediately. The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Return Value The **pop_rset** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_FAILURE

Host was not able to execute command (POP 3 only).

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data (POP 3 only).

POP_TIMED_OUT

Time out occurred while waiting for a response from host (POP 3 only).

POP_IN_PROGRESS

Another operation is already in progress.

POP_ABORTED

Operation was aborted and connection has been closed.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.12 pop_stat ()

Description Get statistics about available messages.

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

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

```
int WINAPI pop_stat (hConn, msgs, bytes)
```

```
    GLOBALHANDLE hConn; Handle of POP structure from pop_(x)open
```

```
    int far *msgs; Buffer for message count
```

```
    unsigned long far *bytes; Buffer for byte count
```

Remarks The **pop_stat** function retrieves the number of messages and their total size in bytes available in the mail box on the POP server. The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Note that this command will not work properly on a POP 2 connection after the **pop_delete** function has been called one or more times.

Return Value The **pop_stat** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_FAILURE

Host was not able to execute command.

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data.

POP_TIMED_OUT

Time out occurred while waiting for a response from host.

POP_IN_PROGRESS

Another operation is already in progress.

POP_ABORTED

Operation was aborted and connection has been closed.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.13 pop_uniq ()

Description Unique message id.

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

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

```
int WINAPI pop_uniq (hConn, msg, id, proc)
```

GLOBALHANDLE <i>hConn</i> ;	Handle of POP structure from pop_(x)open
int <i>msg</i> ;	Message number
char far * <i>id</i> ;	Return buffer
FARPROC <i>proc</i> ;	Application callback function

Remarks The **pop_uniq** function returns the unique-id listing for a given message if the message is not marked as deleted. The unique-id of a message is an arbitrary server-determined string which uniquely identifies a message within a mailbox and persists across all sessions. This command is only supported on a POP 3 connection. On a POP 2 connection, this command returns an error.

The message number is specified in the *msg* parameter and can be set to 0. If the *msg* parameter contains a valid message number, the POP3 server will return a line containing the information for the message. The **pop_uniq** function will return this response to the application in the buffer pointed to by the *id* parameter.

If a message number is not specified (i.e., if *msg* parameter is 0), the POP3 server will return a multi-line response. This response is passed onto the application through the provided callback function *proc*. For each line of response, the callback function *proc* is called with a string specifying the response and the length of the response in bytes. The callback function *proc* must be defined as follows:

```
int WINAPI (*proc) (hConn, line, len, lParam)

GLOBALHANDLE hConn;
char far *line;
int len;
DWORD lParam;
```

An application can abort the response by returning -1 in the callback function (all other return values are currently ignored) or by calling **pop_xquit**. Note that aborting a unique-id listing will also abort the underlying connection by resetting it. If the listing was aborted successfully, then the **pop_uniq** call will return POP_ABORTED and the connection handle *hConn* will no longer be valid. No more calls (including **pop_quit**) can be made until a new connection has been established with **pop_(x)open**.

The connection handle *hConn* must have been obtained with a call to **pop_(x)open**.

Return Value The **pop_uniq** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

Invalid connection handle was specified.

POP_FAILURE

Host was not able to execute command.

POP_IN_PROGRESS

Another operation is already in progress.

POP_ABORTED

Operation was aborted and connection has been closed.

If this function returns POP_FAILURE, then the **pop_rply** function can be used to retrieve the error message sent by the host which describes the reason why the command failed.

2.14 pop_xopen ()

Description Open asynchronous connection with POP server.

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

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

```
int WINAPI pop_xopen (host, type, usr, pwd, conn, dbg, stat, lParam)
```

LPSTR <i>host</i> ;	Name of POP server
int <i>type</i> ;	Type of POP server
LPSTR <i>usr</i> ;	User name
LPSTR <i>pwd</i> ;	Password
GLOBALHANDLE far * <i>conn</i> ;	Buffer for handle of POP structure
FARPROC <i>dbg</i> ;	Application debug callback function
FARPROC <i>stat</i> ;	Application status callback function
DWORD <i>lParam</i> ;	Application supplied parameter

Remarks The **pop_xopen** function establishes a TCP/IP connection with the specified host in an asynchronous manner. The *host* parameter can point either to a machine name (possibly including a domain name) or to an internet address in dotted decimal notation (e.g. "192.1.2.3"). The type of POP server must be specified by setting *type* to either one of the following two values.

TYPE_POP2	POP version 2
TYPE_POP3	POP version 3

Most servers support the more advanced POP 3 protocol. Once the server type has been set, all remaining functions in the Distinct POP-32 library will work without specifying the version of POP.

Unlike the **pop_open** function, the **pop_xopen** function does not actually establish the connection before returning to the application. Instead, it allocates required local resources and returns the handle of the connection control structure in the buffer pointed to by *conn*. Once the connection has been established, the status callback function specified with *stat* will be called. This function must be defined as follows.

```
int WINAPI (*stat) (hConn, result, lParam)

GLOBALHANDLE hConn;
int result;
DWORD lParam;
```

If the connection was established successfully, then the *result* parameter will contain the value POP_SUCCESS. If the connection could not be established, then the *result* parameter will contain one of the following errors.

POP_FAILURE	Invalid user name or password specified.
POP_UNKNOWN_HOST	Unable to resolve specified host name.
POP_CANNOT_ALLOC_SOCKET	Windows Sockets library could not allocate a socket.
POP_CANNOT_BIND_SOCKET	Windows Sockets library was not able to bind socket to local port.
POP_HOST_NOT_RESPONDING	Unable to connect to POP port on specified host.

POP_CANNOT_SEND_COMMAND

Windows Sockets library was not able to accept outgoing data.

POP_TIMED_OUT

Time out occurred while waiting for a response from host.

POP_ABORTED

Asynchronous connection attempt aborted with call to **pop_xquit**.

Once a connection has been established, the caller is automatically logged into the POP server with the user name *usr* and the password *pwd*.

Once the status callback function has been called with *result* set to POP_SUCCESS, then the handle placed into the buffer pointed to by *conn* must be used in all subsequent calls to the other POP functions for this connection. Any calls made with this handle before the connection has been established will fail with a return value of POP_IN_PROGRESS.

Multiple POP connections can be established simultaneously with the **pop_xopen** or **pop_open** calls. The Distinct POP-32 library supports multiple concurrent POP connections limited only by system resources.

During development it may be necessary to observe exactly what commands are sent to the POP server and what replies are generated. For this purpose, the application can specify the address of a debug callback function with *dbg*. This function will receive a copy of all network traffic and must be defined as follows.

```
int WINAPI (*dbg) (hConn, buf, len, lParam)
```

```
GLOBALHANDLE hConn;
char far *buf;
int len;
DWORD lParam;
```

The *dbg* parameter must be set to NULL if the application does not want to receive debug messages.

The *lParam* parameter can be set to any value that has meaning to the application such as an index into an array or a pointer to a structure. This parameter is passed back to the application as an argument in each callback function and allows the application to pass connection specific data to the callback function.

Return Value The **pop_xopen** function either returns POP_SUCCESS or one of the following errors.

POP_CANNOT_INITIALIZE

Support libraries or network drivers could not be initialized.

POP_CANNOT_INIT_WINSOCK

Windows Sockets transport library could not be initialized.

POP_OUT_OF_MEMORY

Not enough memory to allocate connection control structure.

2.15 pop_xquit ()

Description Cancel asynchronous connect or operation in progress.

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

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

```
int WINAPI pop_xquit (hConn)
```

 GLOBALHANDLE *hConn*; Handle of POP structure from **pop_(x)open**

Remarks The **pop_xquit** function terminates an asynchronous connection attempt initiated with a call to **pop_xopen** or a POP operation in progress and frees all associated resources. The connection handle *hConn* must have been obtained with a call to **pop_(x)open** and will be invalid once the **pop_xquit** function returns.

If **pop_xquit** is called to cancel an asynchronous connect, then after the **pop_xquit** call returns, the application must not terminate until the status callback function *stat* specified in the **pop_xopen** call has been called with the *result* parameter set to POP_ABORTED. If **pop_xquit** is called to cancel an operation in progress, then after the **pop_xquit** call returns, the application must not terminate until the canceled function returns.

The **pop_xquit** function can be called after calling **pop_xopen** and before the status callback function *stat* specified in the **pop_xopen** call is called with the result of the connection attempt. The **pop_xquit** function can also be called to cancel a **pop_dele**, **pop_list**, **pop_retr**, **pop_rset** or **pop_stat** operation in progress.

Return Value The **pop_xquit** function either returns POP_SUCCESS or one of the following errors.

POP_INVALID_HANDLE

 Invalid connection handle was specified.

POP_FAILURE

 No asynchronous connect in progress or asynchronous connect already aborted.

