

R-32 Library

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

Version 5.2

**Copyright © 1993 - 2003 by Distinct Corporation
All rights reserved**

Table of Contents

1 Overview	5
1.1 Introduction	5
1.2 When you must be a Trusted Host	6
1.3 Registry Entries	7
1.4 Function Summary	8
2 Reference	9
2.1 rep ()	9
2.2 rexec ()	13
2.3 rlogin ()	15
2.4 rsh ()	16

1 Overview

1.1 Introduction

The Distinct R-32 (Remote Utilities) library provides developers with an API to integrate remote shell (**rsh**), remote login (**rlogin**), remote execution (**rexec**) and remote copy (**rcp**) capabilities into their applications. This library takes care of establishing the connection and any login procedures that may be required for the execution of the command.

These functions require the remote machine to be configured for remote command or remote copy execution. Each remote host must have a file `/etc/hosts.equiv` containing an equivalence list of remote hosts and user names. Each user may also have a private equivalence list in a file `.rhosts` in their login directory. If the necessary permissions are not set on the remote host, then the command will not be executed.

The **rcp** function copies files between two machines. This function does not prompt for passwords. Instead, the specified user name must exist on the remote machine and must be set up to allow command execution via **rsh**. Options to transfer directories recursively and to preserve the date and time of the source files are available. The **rcp** function is meant to copy files between hosts. An attempt to copy a file to itself will result in a corrupted file.

The **rexec** function connects to the specified remote host to execute a given command. User name and password are specified to login to the remote host. If the connection succeeds, then the command is executed and the resulting output is returned to the application. The **rexec** function normally terminates when the remote command has finished executing.

The **rlogin** function connects your application on the local system to the remote host. While logging in, the remote machine will prompt for a user name and password just like during a standard Telnet login. The remote terminal type is the same as your local terminal type. All echoing takes place on the remote system so that (except for network delays) the **rlogin** procedure is transparent. The **rlogin** function does not terminate after the execution of the function. Instead, it allows the application to keep executing commands. This can be used to execute interactive commands such as `vi`.

The **rsh** function connects to the shell utility of the remote host to execute the specified command. No provision is made for specifying the password. The **rsh** function normally terminates when the remote command is executed completely. Shell meta characters can be used as part of the command (for example, `cat test >> test1` appends `test` to `test1`).

An application using the **rsh**, **rlogin** or **rexec** function must call the Windows Sockets library to read the incoming data, send data to the remote server and to close the connection. Before using the **rsh**, **rlogin** or **rexec** function, an application must initialize the Windows Sockets library by calling the **WSAStartup ()** function. Also, an application must shutdown the Windows Sockets library by calling the **WSACleanup ()** function before quitting.

1.2 When you must be a Trusted Host

In order to make a connection using RSH, REXEC or RCopy you must register the PC running your application as a trusted host on the Unix machine that you are trying to connect to.

Here is an example of how to make your PC a Trusted Host:

Assuming you are trying to connect to a UNIX system called `unix1.mydomain.com` (192.0.0.1) and your pc is called `mypc.mydomain.com` (192.0.0.3) you must update the following files to be able to use REXEC, RSH or RCopy successfully:

Add the following entries in `/etc/hosts` file on the Unix system called `unix1.mydomain.com`:

```
192.0.0.3 mypc mypc.mydomain.com
```

Add the following entries in the `/etc/hosts.equiv` file:

```
mypc
mypc.mydomain.com
```

Updating the `/etc/hosts` file allows a reference of the remote host by name. In some cases this file may NOT be read at all. If this is your case, you will have to update the appropriate hosts database file on that system. For example if the host uses a name server you will have to modify the name server's database to achieve the desired result.

If on the other hand you need to run RCopy between two UNIX systems, you will need to do all of the above plus: When running the RCopy command, both UNIX systems must have the other system defined in the hosts file and hosts.equiv files. So if your second system is called `unix2.mydomain.com`, add the following entry in `/etc/hosts` file

```
192.0.0.1 unix1 unix1.mydomain.com
192.0.0.3 mypc mypc.mydomain.com
```

Add the following entries in the `/etc/hosts.equiv` file:

```
unix1
unix1.mydomain.com
mypc
mypc.mydomain.com
```

In this case you will also need to add the information for `unix2.mydomain.com` to the hosts and hosts.equiv files on the `unix1.mydomain.com` system.

1.3 Registry Entries

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

HKEY_LOCAL_MACHINE\SOFTWARE\Distinct\DLLS\RLIB32

The following entries specify various Distinct R-32 library settings. Some parameters are specific to each function whereas others are common to all the functions in the Distinct R-32 library.

RexecTimeout **REG_DWORD** *1-999*

Specifies the default timeout in seconds used while connecting to an rexec server. The default value is 20 seconds.

RexecLog **REG_DWORD** *1-5*

Specifies the queue size. The default value is 5.

RexecSendBuf **REG_DWORD** *1024-32000*

Specifies the size of the send buffer. The default value is 32000 bytes.

RexecReceiveBuf **REG_DWORD** *1024-32000*

Specifies the size of the receive buffer. The default value is 32000 bytes.

RloginTimeout **REG_DWORD** *1-999*

Specifies the default timeout in seconds used while connecting to an rlogin server. The default value is 20 seconds.

RloginSendBuf **REG_DWORD** *1024-32000*

Specifies the size of the send buffer. The default value is 4096 bytes.

RloginReceiveBuf **REG_DWORD** *1024-32000*

Specifies the size of the receive buffer. The default value is 4096 bytes.

RshTimeout **REG_DWORD** *1-999*

Specifies the default timeout in seconds used while connecting to an rsh server. The default value is 20 seconds.

RshLog **REG_DWORD** *1-5*

Specifies the queue size. The default value is 5.

RshSendBuf **REG_DWORD** *1024-32000*

Specifies the size of the send buffer. The default value is 4096 bytes.

RshReceiveBuf **REG_DWORD** *1024-32000*

Specifies the size of the receive buffer. The default value is 4096 bytes.

RcpTimeout **REG_DWORD** *1-999*

Specifies the default timeout in seconds used while connecting to an rcp server. The default value is 20 seconds.

RcpErrorBuffer **REG_DWORD** *128-32000*

Specifies the size of the error buffer. The default value is 2048 bytes.

RcpMaxErrors **REG_DWORD** *1-999*

Specifies the maximum number of errors that are allowed. If the number of errors is greater than this value, then the function will terminate. The default value is 50.

DebugLevel **REG_DWORD** *0-2*

Specifies what debug information should be logged into the D32-RLIB.DBG file in the directory from which D32-RLIB.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.

1.4 Function Summary

rcp

Remote file copy

rexec

Remote execution

rlogin

Remote login

rsh

Remote shell

2 Reference

2.1 rcp ()

Description

Remote file copy.

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

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

```
int WINAPI rcp (host1, user1, file1, host2, user2, file2, errbuf, flags, proc, lParam)
```

LPSTR <i>host1</i> ;	Source host name
LPSTR <i>user1</i> ;	Source user login name
LPSTR <i>file1</i> ;	Source file or directory name
LPSTR <i>host2</i> ;	Destination host name
LPSTR <i>user2</i> ;	Destination user login name
LPSTR <i>file2</i> ;	Destination file or directory name
char far * <i>errbuf</i> ;	Error message return buffer
unsigned int <i>flags</i> ;	Options used during copy
FARPROC <i>proc</i> ;	Application callback function
DWORD <i>lParam</i> ;	Application supplied parameter

Remarks

The **rcp** function allows the application to copy files between any two machines. Files can be copied either between the local and a remote machine or between two remote machines. The destination *file2* can either be a file or a directory whereas the source *file1* can contain one or more file or directory names separated by a space. If *file1* is a single file name then *file2* must also be a single file name.

There are three different types of copy operations.

Local to Remote Copy

Both *host1* and *user1* must be NULL or point to empty strings to indicate that the source is the local machine. The *host2* and *user2* arguments must specify the host name and user name on the remote destination machine. The *file1* argument specifies the source files and *file2* must point to the destination filenames.

Remote to Local Copy

Both *host2* and *user2* must be NULL or point to empty strings to indicate that the destination is the local machine. The *host1* and *user1* arguments must specify the host name and user name on the remote machine containing the source files. The *file2* argument specifies the source files and *file1* must point to the destination filenames.

Remote to Remote Copy

The *host1* and *user1* arguments must specify the host name and user name on the remote machine containing the source files. The *host2* and *user2* arguments must specify the host name and user name on the remote destination machine. The *file1* argument specifies the source files and the *file2* argument must point to the name of the destination directory.

The **rcp** function establishes a connection with the remote host using the **rsh** function. All open connections will be closed before the function returns control to the application. The application must have at least a 16 KB stack to handle recursive file transfers.

If the *errbuf* argument is not NULL, then any error messages returned by the remote machine or by the R-32 library will be copied into the buffer pointed to by *errbuf*. If *errbuf* is used then it must point to a buffer capable of accepting at least 2048 characters. One or more lines of diagnostic error messages may be copied into the buffer pointed to by *errbuf*.

The argument *flags* specifies various options of the **rcp** command. The value of *flags* can be any combination of the constants listed below. The logical OR operator can be used to combine different options.

Value	Meaning
RCP_NONE	No options.
RCP_PRESERVE	Preserve time and date of original files.
RCP_RECURSIVE	Copy all subdirectories recursively.
RCP_EVENT_BASED_COPY	Copying of files will be done by application.

The argument *proc* defines a callback function and must be exported by the application. There are two different types of callback functions.

If the RCP_EVENT_BASED_COPY option is **not** used, then the application can get status information during the copy process by supplying the **rcp** function with a status callback function in *proc*. If *proc* is not NULL, then the **rcp** function will call *proc* with the file information after each copy operation. If *proc* is NULL then the **rcp** function will not return control to the application until the file copy operation has been completed. The status callback function must be defined as follows.

```
int WINAPI (*proc) (rcp_socket, rcpinfo, lParam)

int rcp_socket;           /* connection id */
RCPINFO far *rcpinfo;    /* file information */
DWORD lParam;           /* application supplied parameter */
```

The argument *rcpinfo* will contain information (see below) about the file or directory that has been copied. To abort the copy operation at this point, the application can return -1 from *proc*. This will cause the **rcp** function to reset the connection with the remote machine and return RCP_ABORTED.

If the RCP_EVENT_BASED_COPY option is used, then the **rcp** function will call the application one or more times to deliver or obtain information or data. In this case the argument *proc* can **not** be NULL and the callback function *proc* must be defined as follows.

```
int WINAPI (*proc) (rcp_socket, rcpinfo, buf, len, type, lParam)

int rcp_socket;           /* connection id */
RCPINFO far *rcpinfo;    /* file information */
char far *buf;           /* buffer */
int far *len;            /* length of buffer */
int type;                /* type of transfer */
DWORD lParam;           /* application supplied parameter */
```

On each call, depending on the value of *type*, the application either sends file information or data to the library or receives file information or data from the library. The *type* parameter will contain one of the following constants.

Value	Meaning
TYPE_RECV_INFO	Receive file or directory information.
TYPE_RECV_DATA	Receive file data.
TYPE_SEND_INFO	Send file or directory information.
TYPE_SEND_DATA	Send file data.

TYPE_RECV_INFO

The *rcpinfo* structure (see below) contains the information about the file received from the remote machine.

TYPE_RECV_DATA

The buffer pointed to by *buf* contains all or a portion of the contents of the file being copied from the remote host and the argument *len* specifies the number of bytes in the buffer. If *len* is 0, then the end of the file has been reached. For directories, the application will not receive any TYPE_RECV_DATA callbacks.

TYPE_SEND_INFO

The application must fill the *rcpinfo* structure (see below) with information about the file being sent to the remote host. To indicate that there are no more files or directories to transfer the application must return FALSE.

TYPE_SEND_DATA

The application must copy the next portion of the data to be written to the remote file into the buffer pointed to by *buf* and must set the *len* argument to the number of bytes copied into the buffer. Initially, the *len* parameter is set to indicate the maximum number of bytes that can be copied into the buffer. Setting *len* to 0 indicates that the end of the file has been reached.

To abort the copy operation at any point, the application can return -1 from *proc*. This will cause the **rcp** function to reset the connection with the remote machine and return RCP_ABORTED.

The RCPINFO structure used by the callback functions is defined as follows.

```
typedef struct _rcp_info
{
    LPSTR name;                /* file or directory name */
    BOOL directory;           /* directory (TRUE) or file (FALSE) */
    unsigned long mod_time;    /* modification time */
    unsigned long acc_time;    /* access time */
    unsigned long size;        /* file size (0 for directory) */
    unsigned int mode;         /* attributes for access */
    char reserved [64 - sizeof (LPSTR) - sizeof (BOOL) - (3 * sizeof
(unsigned long)) - sizeof (unsigned int)];
} RCPINFO;
```

The *name* argument points to a buffer containing the name of the current file or directory. If the value of *directory* is TRUE then *name* specifies a directory. If *directory* is FALSE then *name* refers to a file. If the RCP_PRESERVE flag is set then *mod_time* and *acc_time* will contain the modification and access times of the file or directory specified by *name*. The value of the constant TIME_DIFFERENCE must be added to *mod_time* and *acc_time* during TYPE_RECV_INFO upcalls and subtracted during TYPE_SEND_INFO upcalls. The constant TIME_DIFFERENCE is declared in the **d32-rlib.h** header file and is used to convert time values between UNIX and DOS systems. The size of the file is specified by *size* and will be 0 for a directory. The value of *mode* specifies the file permissions and can be any combination of the values listed below. The logical OR operator can be used to combine different values for *mode*.

Value	Meaning
FILE_EXEC	Execute permission.
FILE_WRITE	Write permission.
FILE_READ	Read permission.

The argument *lParam* 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.

Note: At least one of the hosts must be a remote host. This means that both *host1* and *host2* cannot be NULL or point to empty strings. Also, both the source and the destination file names *file1* and *file2* must always be specified.

Return Value If no error occurs then the **rcp** function returns 0. If the return value is -1 then the buffer pointed to by *errbuf* will contain diagnostic messages. Otherwise one of the following error values will be returned.

RCP_ABORTED

File copy was aborted.

INVALID_SOCKET

Unable to allocate a socket.

ENOBUFS

Not enough memory to allocate buffers for copy operation.

2.2 rexec ()

Description Remote execution.

```
#include <windows.h>
#include <d32-rlib.h>

int WINAPI rexec (host, port, usr, pwd, cmd, fd2p, stdio_msg, aux_msg, errbuf, hWin)

LPSTR host;           Remote host name
unsigned int port;    Local port number
LPSTR usr;           User login name
LPSTR pwd;           User login password
LPSTR cmd;           Command to execute
int far *fd2p;       Auxiliary port for error messages
unsigned int stdio_msg; Upcall message for data on local port
unsigned int aux_msg; Upcall message for data on auxiliary port
char far *errbuf;    Error message return buffer
HWND hWin;          Handle of window which will process upcalls
```

Remarks The **rexec** function establishes a connection with the remote host given by *host*. The *host* argument can specify either a machine name or a dotted decimal internet address. If the *port* argument is set to 0 then a default port will be used as the local port. If a user name and a password are specified with the *usr* and *pwd* arguments, then they will be used for authentication during the login process. If a password will not be required, then the *pwd* argument can be set to NULL or point to an empty string.

The command to be executed once a connection has been established is specified by the *cmd* argument. If the argument *fd2p* points to a nonzero value then this value will be used as the port number for the auxiliary socket. An auxiliary channel will be set up and the value of the auxiliary socket descriptor will be returned in the buffer pointed to by *fd2p*. This channel will receive any diagnostic messages from the remote machine. If *fd2p* points to the constant NO_AUX_PORT then the auxiliary port is not used.

The *hWin* argument specifies the handle of the window which will receive the upcall messages *stdio_msg* and *aux_msg* whenever data arrives on the local or auxiliary port. If *fd2p* points to the constant NO_AUX_PORT then the argument *aux_msg* is ignored. If *hWin* is set to NULL, then the upcall messages *stdio_msg* and *aux_msg* are both ignored.

Once a connection has been established a socket descriptor is returned to the application. The results of the execution of the command *cmd* will be received by the application on this socket. The upcall messages *stdio_msg* and *aux_msg* will be used to notify the application of any data arrival. The socket descriptor returned by the **rexec** function and the optional socket descriptor placed into the buffer pointed to by *fd2p* must be closed by the application before quitting.

If the *errbuf* argument is not NULL, then any diagnostic messages related to connection failure or command execution failure returned by the remote machine will be copied into the buffer pointed to by *errbuf*. If *errbuf* is used then it must point to a buffer capable of accepting at least 128 characters.

Return Value If no error occurs then the **rexec** function returns a socket descriptor. If the return value is -1 then the buffer pointed to by *errbuf* will contain diagnostic messages. If the auxiliary connection could not be established, then the return value will be -1 and -1 will also be placed into the buffer pointed to by *fd2p*. Otherwise one of the following error values will be returned.

INVALID_SOCKET

Unable to allocate a socket.

ENOBUFS

Not enough memory to buffer request.

2.3 rlogin ()

Description Remote login.

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

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

```
int WINAPI rlogin (host, port, usr, msg, errbuf, hWin)
```

LPSTR <i>host</i> ;	Remote host name
unsigned int <i>port</i> ;	Local port number
LPSTR <i>usr</i> ;	User login name
unsigned int <i>msg</i> ;	Upcall message for incoming data
char far * <i>errbuf</i> ;	Error message return buffer
HWND <i>hWin</i> ;	Handle of window which will process upcalls

Remarks The **rlogin** function establishes a connection with the remote host given by *host*. The *host* argument can specify either a machine name or a dotted decimal internet address. If the *port* argument is set to 0 then a default port will be used as the local port. A user name must be specified with the *usr* argument to complete the login process. A password is not required.

The *hWin* argument specifies the handle of the window which will receive the upcall message *msg* whenever data arrives on the local port. If *hWin* is NULL then the upcall message *msg* is ignored.

Once a connection has been established a socket descriptor is returned to the application. The application can then send one or more commands to be executed to the remote host. The results of any such subsequent commands sent to the remote machine will be received on this socket. The connection will stay open until a command to disconnect is executed on the remote machine or the socket is closed locally. The upcall message *msg* will be used to notify the application of any data arrival. The socket descriptor returned by the **rlogin** function must be closed by the application before quitting.

If the *errbuf* argument is not NULL, then any diagnostic messages related to connection failure or command execution failure returned by the remote machine will be copied into the buffer pointed to by *errbuf*. If *errbuf* is used then it must point to a buffer capable of accepting at least 128 characters.

RFC 1282 - "BSD Rlogin" describes this protocol in detail.

Return Value If no error occurs then the **rlogin** function returns a socket descriptor. If the return value is -1 then the buffer pointed to by *errbuf* will contain diagnostic messages. Otherwise one of the following error values will be returned.

INVALID_SOCKET

Unable to allocate a socket.

ENOBUFS

Not enough memory to buffer request.

2.4 rsh ()

Description Remote shell.

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

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

```
int WINAPI rsh (host, port, usr, cmd, fd2p, stdio_msg, aux_msg, errbuf, hWin)
```

LPSTR <i>host</i> ;	Remote host name
unsigned int <i>port</i> ;	Local port number
LPSTR <i>usr</i> ;	User login name
LPSTR <i>cmd</i> ;	Command to execute
int far * <i>fd2p</i> ;	Auxiliary port for error messages
unsigned int <i>stdio_msg</i> ;	Upcall message for data on local port
unsigned int <i>aux_msg</i> ;	Upcall message for data on auxiliary port
char far * <i>errbuf</i> ;	Error message return buffer
HWND <i>hWin</i> ;	Handle of window which will process upcalls

Remarks The **rsh** function establishes a connection with the remote host given by *host*. The *host* argument can specify either a machine name or a dotted decimal internet address. If the *port* argument is set to 0 then a default port will be used as the local port. A user name must be specified with the *usr* argument to complete the login process. A password is not required.

The command to be executed once a connection has been established is specified by the *cmd* argument. Shell meta characters must be specified within quotes to be interpreted correctly on the remote machine.

If the argument *fd2p* points to a nonzero value then this value will be used as the port number for the auxiliary socket. An auxiliary channel will be set up and the value of the auxiliary socket descriptor will be returned in the buffer pointed to by *fd2p*. This channel will receive any diagnostic messages from the remote machine. If *fd2p* points to the constant NO_AUX_PORT then the auxiliary port is not used.

The *hWin* argument specifies the handle of the window which will receive the upcall messages *stdio_msg* and *aux_msg* whenever data arrives on the local or auxiliary port. If *fd2p* points to the constant NO_AUX_PORT then the argument *aux_msg* is ignored. If *hWin* is set to NULL, then the upcall messages *stdio_msg* and *aux_msg* are both ignored.

Once a connection has been established a socket descriptor is returned to the application. The results of the execution of the command *cmd* will be received by the application on this socket. The upcall messages *stdio_msg* and *aux_msg* will be used to notify the application of any data arrival. The socket descriptor returned by the **rsh** function and the optional socket descriptor placed into the buffer pointed to by *fd2p* must be closed by the application before quitting.

If the *errbuf* argument is not NULL, then any diagnostic messages related to connection failure or command execution failure returned by the remote machine will be copied into the buffer pointed to by *errbuf*. If *errbuf* is used then it must point to a buffer capable of accepting at least 128 characters.

Return Value If no error occurs then the **rsh** function returns a socket descriptor. If the return value is -1 then the buffer pointed to by *errbuf* will contain diagnostic messages. If the auxiliary connection could not be established, then the return value will be -1 and -1 will also be placed into the buffer pointed to by *fd2p*. Otherwise one of the following error values will be returned.

INVALID_SOCKET

Unable to allocate a socket.

ENOBUFS

Not enough memory to buffer request.