

**Distinct<sup>®</sup>**  
**WebTelnet**  
**Objects<sup>™</sup>**  
**Programmer's**  
**Guide**

**Version 1.0**

## **Distinct Corporation**

3315 Almaden Expressway  
San Jose, CA 95118 USA

Phone: +1 408-445-3270

Fax: +1 408-445-3284

Email: sales@distinct.com

WWW: <http://www.distinct.com>

### **Disclaimer**

Distinct Corporation makes no warranties as to the contents of this documentation and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Information in this manual is subject to change without notice and does not represent a commitment on the part of Distinct Corporation. The Software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement.

### **Copyright Notice**

© 1999 - 2001 by Distinct Corporation. All rights reserved.

No part of this publication may be reproduced, transmitted or translated into any language by any means without the express written permission of Distinct Corporation.

### **Trademarks**

Distinct is a registered trademark and WebTernet Objects is a trademark of Distinct Corporation. Windows is a registered trademark of Microsoft Corporation. Other product names are trademarks or registered trademarks of their respective owners.

October 15th, 2001

Published in the United States of America

---

# Table of Contents

TABLE OF CONTENTS .....	III
<b>1 - INTRODUCTION.....</b>	<b>1</b>
OVERVIEW .....	1
<i>About this manual.....</i>	<i>1</i>
<b>2 - INSTALLATION .....</b>	<b>3</b>
INSTALLING WEBTELNET OBJECTS .....	3
<i>Uninstalling WebTelnet Objects.....</i>	<i>3</i>
<i>Sample Applications .....</i>	<i>3</i>
<b>3 – TELNET OBJECT .....</b>	<b>5</b>
TELNET OVERVIEW .....	5
<i>Including the header file in your ASP or HTML page.....</i>	<i>5</i>
<i>Adding the Telnet Object to your HTML Page.....</i>	<i>5</i>
<i>Adding the Object to your ASP Page.....</i>	<i>6</i>
<i>Automated Logins.....</i>	<i>6</i>
Registry Entries .....	7
TELNET PROPERTIES .....	8
Binary.....	9
Command Prompt.....	10
Echo.....	11
LoginPrompt.....	13
Password Prompt .....	14
SocksHostAddrType.....	15
SocksAuthMethods.....	16
SocksPassword .....	17
SocksUsername.....	18
SocksVer.....	19
SocksServer .....	20
SocksPort.....	21
Port.....	22
ReceiveData.....	23
ReceiveCount.....	24
ReceiveInFile.....	25
ReceivedDataPath.....	26
ReceivedDataFileName .....	28
TELNET METHODS .....	29
Abort ().....	30
Connect ().....	31
Disconnect ().....	33
Receive ().....	34
Send ().....	36
<b>4 – REMOTE COPY OBJECT .....</b>	<b>39</b>
REMOTE COPY OVERVIEW .....	39
<i>What needs to be set up on the remote system.....</i>	<i>39</i>
<i>How to Use the RCP COM Object .....</i>	<i>39</i>
How to embed the control in a HTML page .....	40
How to embed the control in an ASP page .....	40
Registry Entries .....	40
RCP PROPERTIES .....	41
DestFile .....	42

<i>DestHost</i> .....	45
<i>DestUser</i> .....	47
<i>ErrorMessage</i> .....	48
<i>FileAttributes</i> .....	49
<i>FileDate</i> .....	50
<i>FileName</i> .....	51
<i>FileSize</i> .....	52
<i>FileTime</i> .....	53
<i>FileType</i> .....	54
<i>Options</i> .....	55
<i>SrcFile</i> .....	57
<i>SrcHost</i> .....	58
<i>SrcUser</i> .....	59
<b>RCP METHODS</b> .....	<b>60</b>
<i>Abort ()</i> .....	61
<i>FileCopy</i> .....	62
<b>5 – REMOTE COMMANDS OBJECT</b> .....	<b>65</b>
<b>REMOTE COMMANDS OVERVIEW</b> .....	<b>65</b>
<i>You must be a Trusted Host to use RSH</i> .....	65
<i>How to Use the Remote Commands Object</i> .....	66
How to embed the control in a HTML page:.....	66
How to embed the control in an ASP page:.....	66
Registry Entries.....	66
<b>REMOTE COMMANDS PROPERTIES</b> .....	<b>68</b>
<i>CommandCompleted</i> .....	69
<i>ErrorMessage</i> .....	71
<i>ReceivedData</i> .....	72
<i>ReceivedDataFileName</i> .....	74
<i>ReceivedDataPath</i> .....	76
<i>ReceiveInFile</i> .....	77
<b>REMOTE OBJECTS METHODS</b> .....	<b>78</b>
<i>Abort ()</i> .....	79
<i>Rexec ()</i> .....	80
<i>Receive ()</i> .....	82
<i>Rsh ()</i> .....	84
<b>6 – RLOGIN OBJECT</b> .....	<b>86</b>
<b>RLOGIN OVERVIEW</b> .....	<b>86</b>
<i>How to Use the Rlogin Object</i> .....	86
How to embed the control in a HTML page.....	86
How to embed the control in an ASP page.....	87
Registry Entries for Rlogin.....	87
<b>RLOGIN PROPERTIES</b> .....	<b>88</b>
<i>CmdlinePrompt</i> .....	89
<i>ErrorMessage</i> .....	90
<i>Password Prompt</i> .....	91
<i>ReceivedData</i> .....	92
<i>ReceivedDataPath</i> .....	93
<i>ReceivedDataFileName</i> .....	94
<i>ReceiveInFile</i> .....	95
<b>RLOGIN METHODS</b> .....	<b>96</b>
<i>Abort ()</i> .....	97
<i>Disconnect</i> .....	98
<i>Receive ()</i> .....	99
<i>Rlogin</i> .....	100



# 1 - Introduction

## Overview

Distinct WebTelnet Objects contains Objects for remote login and command execution. The product covers the Telnet protocol and the RCP (Remote Copy), RSH (Remote Shell), REXEC (Remote Execution) and Rlogin (Remote Login) commands. The choice of protocol or command to use largely depends on the server setup and permission levels for remote login and execution.

## About this manual

This manual contains all the information that Web developers need to integrate Telnet, Rlogin, RSH, REXEC or RCP capabilities into their ASP and HTML pages. Chapter 3 discusses the Telnet API, Chapter 4 covers Remote Copy (RCP), Chapter 5 covers REXEC and RSH and chapter 6 Rlogin.



## 2 - Installation

### Installing WebTelnet Objects

To install the package simply run WebTelnet Objects.exe on your system.

When the installation requests your serial number, enter the license number and keycode that came with your package. If you are running a trial copy of WebTelnet Objects please select TRIAL, this will enable the 15-day trial on your system.

To upgrade your system from a TRIAL copy to a licensed copy of Distinct Telnet Objects you must first purchase a license. Your license will come with a serial number and keycode. Run the WebTelnet Objects License Manager from the Programs/Distinct group and enter the number when prompted.

### Uninstalling WebTelnet Objects

To uninstall the package simply go to Add/Remove programs and select WebTelnet Objects and uninstall it in the normal way.

### Sample Applications

A number of samples were installed on your system. The samples folder contains subfolders for Telnet, Rcmd, Rcopy and Rlogin. Each of these folders contains samples for the respective protocols.



---

## 3 – Telnet Object

### Telnet Overview

The Telnet Object allows you to make a connection to the Telnet server and send and receive data. It specifically allows automated Telnet logins. The Telnet protocol provides a way for a system to remotely log into a remote system. Once the login session is established the client will pass the user's keystrokes to the server process.

### Including the header file in your ASP or HTML page

To simplify your scripting you can include the header files where the constants are defined. The header files are installed in a subfolder called Headers. There are two header files per Object, one for JAVA script and one for VB script. To use these header files on the server side you must enclose the file contents in `<%...%>`. To include the header file for each WebTelnet Object you are using you need to add one of the following HTML tags to your page:

#### On the IIS server for ASP pages:

```
<!--#include virtual = "relative path under web root/filename">
```

where *relative path under web root* is the path starting from the web root where you have installed the header files and *filename* is the name of the specific header file to include.

Or

```
<!--#include file = "relative path starting from the current directory">
```

Note that the header file used must contain `<%` at the start of the file and `%>` at the end of the file.

#### For HTML pages that will be downloaded on the client:

To include the header file in an HTML (client side) page the `<SCRIPT>` tag may be used as follows:

```
<SCRIPT SRC="relative path to file"></SCRIPT>
```

where path is relative to the current directory (that contains the current html file).

### Adding the Telnet Object to your HTML Page

The Telnet object can be put on an html page using the following `<Object>` tags:

```
<OBJECT ID="Telnet1" WIDTH="1" HEIGHT="1"  
  CLASSID=" CLSID:BC9F662E-E777-43BE-B54C-5C77CCD537F7">  
  CODEBASE="./dsv_rcp.cab">  
</OBJECT>
```

Note that you need to create a cab file that contains the `dsv_tnet.ocx`, `dsv-tnet.dll` and `dsv-fw.dll` files as shown in the sample code that comes with this product.

## Adding the Object to your ASP Page

When using VB Script on an ASP page add the Object using CreateObject:

```
SET TelnetObj = Server.CreateObject ("DistinctServerTelnet.TelnetCtrl")
```

## Automated Logins

The following table illustrates the behavior of the Telnet Object when an automated login is attempted. An N in the table indicates that the respective property has not been set, and a Y indicates that it has.

LoginPrompt	User	PwdPrompt	Pwd	CmdPrompt	Action
N	N	N	N	N	Connect, return
N	N	N	N	Y	Connect, wait for CmdPrompt
N	N	N	Y	N	Error
N	N	N	Y	Y	Error
N	N	Y	N	N	Connect, wait for PwdPrompt
N	N	Y	N	Y	Connect and wait for PwdPrompt
N	N	Y	Y	N	Connect, wait for PwdPrompt, send password
N	N	Y	Y	Y	Connect, wait for PwdPrompt, send pwd, wait for CmdPrompt
N	Y	N	N	N	Error, Login Prompt missing
N	Y	N	N	Y	Error, Login Prompt missing
N	Y	N	Y	N	Error, Login Prompt missing
N	Y	N	Y	Y	Error, Login Prompt missing
N	Y	Y	N	N	Error, Login Prompt missing
N	Y	Y	N	Y	Error, Login Prompt missing
N	Y	Y	Y	N	Error, Login Prompt missing
N	Y	Y	Y	Y	Error, Login Prompt missing
Y	N	N	N	N	Connect, wait for Login prompt
Y	N	N	N	Y	Connect, wait for login prompt
Y	N	N	Y	N	Connect, wait for login prompt
Y	N	N	Y	Y	Connect, wait for login prompt
Y	N	Y	N	N	Connect, wait for login prompt
Y	N	Y	N	Y	Connect, wait for login prompt
Y	N	Y	Y	N	Connect, wait for login prompt
Y	N	Y	Y	Y	Connect, wait for login prompt
Y	Y	N	N	N	Connect, wait for login prompt, send login, return
Y	Y	N	N	Y	Connect, wait for login prompt, send login, wait for cmd prompt
Y	Y	N	Y	N	Error, pwd prompt missing
Y	Y	N	Y	Y	Error, pwd prompt missing
Y	Y	Y	N	N	Connect, wait for login prompt, send login, wait for pwd prompt
Y	Y	Y	N	Y	Connect, wait for login prompt, send login, wait for pwd prompt
Y	Y	Y	Y	N	Connect, wait for login prompt, send login, wait for password prompt, send password, return

## Distinct Telnet COM Object

Y	Y	Y	Y	Y	Connect, wait for login prompt, send login, wait for password prompt, send password, wait for cmd prompt
---	---	---	---	---	--

## Registry Entries

The following registry entries may be used to modify the behavior of the Telnet Object.

Create registry entries under the key:

HKLM\Software\Distinct\Dlls\Web\Telnet

Timeout REG\_DWORD 1-999

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

BufferSize REG\_DWORD 128-8192

This changes the size of the buffer used to store Telnet option negotiation commands. The default value is 1024 bytes.

MaxConnections REG\_DWORD *number*

This specifies the total number of concurrent Telnet sessions permitted. The default value is 16.

---

## Telnet Properties

The following is a description of all the properties in the Telnet COM Object.

### Property Summary

The following lists all the properties of the Telnet COM Object and gives a brief summary of their function:

Property	Function
<a href="#">Binary</a>	Enables or disables binary mode.
<a href="#">CommandPrompt</a>	Contains the expected command prompt.
<a href="#">Echo</a>	Controls how local input is echoed back.
<a href="#">LoginPrompt</a>	Contains the string value of the expected login prompt.
<a href="#">PasswordPrompt</a>	Contains the expected password prompt.
<a href="#">SocksHostAddrType</a>	Specifies the type of address used to connect through SOCKS.
<a href="#">SocksAuthMethod</a>	Specifies the Authentication method to be used to connect to the SOCKS server.
<a href="#">SocksPassword</a>	Specifies the password to be used when connecting to the SOCKS server.
<a href="#">SocksVer</a>	Specifies the SOCKS server version.
<a href="#">SocksUserName</a>	Specifies the account name on the SOCKS server that will be used to make the connection.
<a href="#">SocksServer</a>	Socks server to be used when connecting through SOCKS.
<a href="#">SocksPort</a>	Socks port through which the connection will be made.
<a href="#">Port</a>	The remote Telnet port to connect to.
<a href="#">ReceivedData</a>	ASCII data that has been received
<a href="#">ReceiveCount</a>	Contains the number of bytes of data that are ready to be read from the socket.
<a href="#">ReceiveInFile</a>	Must be set to store all the incoming data to a file.
<a href="#">ReceivedDataPath</a>	Sets the path to store the data file if the data is being saved to file.
<a href="#">ReceivedDataFileName</a>	Contains the name of the file that the data was saved to.

---

## Binary

---

### Summary:

When set to true, the Telnet client will issue a request to start transmitting in binary mode.

### Description:

When this property is set to true, the Telnet client will try to negotiate Binary mode transmission with the server. The server may accept or reject this request.

### Note:

With the binary transmission option in effect, the receiver should interpret characters received from the transmitter which are not preceded with IAC as 8 bit binary data, with the exception of IAC followed by IAC which stands for the 8 bit binary data with the decimal value 255. IAC followed by an effective TELNET command (plus any additional characters required to complete the command) is still the command even with the binary transmission option in effect.

Once the connection is made in binary mode, if the client needs to issue any further Telnet commands it must do this by placing them in a file and using the Send method to do this.

### Type:

Boolean

### Default Value:

The default value for this property is false.

### See Also:

ReceiveInFile, ReceivedDataPath and ReceivedDataFileName properties.

### Example:

```
//this connects to the server and automatically logs in the user.
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";

    // connect
    Telnet1.Binary = True;
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## Command Prompt

---

### Summary:

May be set to several possible server command prompts.

### Description:

This property is used to recognize the server command prompt when making a connection. The Telnet Object will wait for a command prompt that matches one of the entries in this property before returning. This property can contain several alternative prompts separated by commas. See the section on automated logins in the Introduction for a detailed description of what action the object will take under various scenarios.

### Type:

String

### Default Value:

%,%

### Warning:

The values of these prompts are case sensitive.

### See Also:

LoginPrompt and PasswordPrompt properties.

### Example:

```
//this connects to the server and automatically logs in the user.
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";

    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

## Echo

### Summary:

Requests that either the remote system echoes the data or requests that the remote system does not echo the data.

### Description:

The Echo property controls how local input (any data assigned to the Send property) is echoed back. The property can be set to either one of the following two values.

Value	Meaning
ECHO_REMOTE	Characters are echoed back by the server.
ECHO_LOCAL	The application is responsible to display the characters.

This property must be set before establishing the connection or once a connection is established.

Most Telnet sessions rely on remote echo. This means that any character sent to the server is echoed back to the application. The application can then display the echoed back message to the user. Remote echo allows the server to translate one character to a different sequence of characters. For example, many Telnet servers translate the ^D character (Control - D) to the command "logout". Also, a control character is often interpreted as a cursor control character. For example, entering ^F (Control - F) during a vi session instructs vi to scroll forward by one page. In this case the control character is sent to the server and the server then updates the display with new data without ever echoing back the actual control character.

In some cases, it may be necessary to use local echo. In this case you need to set Echo property to ECHO\_LOCAL. This will cause the server to not echo back any characters it receives. It may, however, still react to certain cursor control characters.

### Type:

Short

### Default Value:

The default value of this property is ECHO\_REMOTE.

### See Also:

Connect () method.

### Example:

```
//this connects to the server and automatically logs in the user.
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";
    Telnet1.Echo = ECHO_LOCAL;
    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
}
```

Distinct Telnet COM Object

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## LoginPrompt

---

### Summary

May be set to several possible server login prompts.

### Description:

When this property is set, the Telnet Object will wait for a login prompt that matches one of the entries in this property before returning. This property can contain several alternative prompts separated by commas. See the section on automated logins in the Introduction for a detailed description of what action the object will take under various scenarios.

### Type:

String

### Warning:

These prompts are case sensitive.

### Default Value:

ogin:,ame:

### See Also:

PasswordPrompt and CommandPrompt properties.

### Example:

```
//this connects to the server and automatically logs in the user.
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";

    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## Password Prompt

---

### Summary:

Contains one or more possible server password prompts.

### Description:

When this property is set, the Telnet Object will wait for a password prompt that matches one of the entries in this property before returning. This property can contain several alternative prompts separated by commas. See the section on automated logins in the Introduction for a detailed description of what action the object will take under various scenarios.

### Type:

String

### Warning:

These prompts are case sensitive.

### Default Value:

word:

### See Also:

LoginPrompt and CommandPrompt properties

### Example:

```
//this connects to the server and automatically logs in the user.
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";

    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## SocksHostAddrType

---

### Summary:

Specifies the Address type of the Telnet server that is to be accessed through the SOCKS proxy server.

### Description:

The *SocksHostAddrType* property specifies the address type for the Telnet server that is to be accessed through the SOCKS proxy server. This property must be set before calling the Connect method when the address is a domain name. The *SocksHostAddrType* property can have two possible values:

Value	Assigned to:	Meaning
1	SOCKS_ADDR_IP4	Address is a version 4 IP address
3	SOCKS_ADDR_DNS	Address is a DNS style domain name

The *SocksHostAddrType* property must be set to SOCKS\_ADDR\_IP4 if the application is specifying an Internet address in the dotted decimal notation (**196.210.123.33**) for the host name of the Telnet server. If the application wants to specify the system name or system name and domain name (for example **speedy.distinct.com**) as the host, the *SocksHostAddrType* property must be set to SOCKS\_ADDR\_DNS.

When changing the default value, this property must be changed before a connection has been established.

### Type:

Short.

### Default Value:

The default value for this property is SOCKS\_ADDR\_IP4.

### See Also:

SocksAuthMethods, SocksUsername, SocksPassword, SocksVer, SocksServer and SocksPort properties and the Connect method

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com
result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## SocksAuthMethods

---

### Summary:

Specifies the SOCKS authentication method for the connection to the SOCKS proxy server.

### Description:

The *SocksAuthMethods* property specifies the authentication method that the client is able to use to connect to the socks server. This property must be set before a connection can be established by calling the Connect method. The *SocksAuthMethods* can be "0" or "2". "0" means that no authentication is required, and "2" means that a valid username and password is required.

This property should be changed before a connection has been established.

### Type:

String

### Default Value:

The default value for this property is "0".

### See Also:

SocksHostAddrType, SocksUsername, SocksPassword, SocksVer, SocksServer and SocksPort properties and the Connect method.

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com
result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## SocksPassword

---

### Summary:

A valid password for the user account authentication by a SOCKS version5 server.

### Description:

The *SocksPassword* property specifies the password for the user account name specified in the *SocksUsername* property that is used to connect to the SOCKS proxy server. A valid password is required when connecting to Socks version 5 server if the authentication method (*SocksAuthMethods*) specified is "2" (Username/Password authentication protocol).

This property should be changed before the Connect method is called to establish a connection.

### Type:

String

### Default Value:

This property has no default value.

### See Also:

*SocksHostAddrType*, *SocksUsername*, *SocksAuthMethod*, *SocksVer*, *SocksServer* and *SocksPort* properties and the *Connect* method.

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com
result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## SocksUsername

---

### Summary:

A valid user name for user account authentication by a SOCKS 5 server.

### Description:

The *SocksUsername* property specifies the name of the user account that is to be used to make an Telnet connection through a SOCKS 5 server by calling the Connect method. This property must be set before the Connect method is called to make a connection through a SOCKS 5 proxy server.

### Type:

String.

### Note:

A valid user id is required when connecting to Socks version 5 server if the authentication method (*SocksAuthMethods*) specified is "2" (Username/Password authentication protocol). It is mandatory when a connection needs to be established with Socks version 5 server.

### Default Value:

This property has no default value.

### See Also:

*SocksHostAddrType*, *SocksPassword*, *SocksAuthMethod*, *SocksVer*, *SocksServer* and *SocksPort* properties and the *Connect* method.

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com

result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## SocksVer

---

### Summary:

This property specifies the SOCKS server version.

### Description:

The *SocksVer* property is used to specify the version of the Socks server. This property can have one of the following values.

Value	Assigned to:	Meaning
2	SOCKS_VERSION5	The Socks version is 5.
4	SOCKS_VERSION4	The Socks version is 4

This property must be set before the Connect method is called to make a connection through a SOCKS proxy server.

### Type:

Short.

### Warning:

If the application is not sure about the SOCKS version then it can specify both SOCKS\_VERSION5 and SOCKS\_VERSION4. The Distinct TELNET object will automatically detect the SOCKS server version and make the appropriate connection.

### See Also:

SocksHostAddrType, SocksPassword, SocksAuthMethod, SocksUsername, SocksServer and SocksPort properties and the Connect method.

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com
result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Property Summary](#)

---

## SocksServer

---

### Summary:

This property determines whether a Telnet connection is made through a SOCKS server.

### Description:

This property should be set to the name or IP address of the SOCKS server. If this property is empty it is assumed that the connection will not be made through a SOCKS server.

### Type:

String

### Default Value:

The default value of this property is empty.

### See Also:

SocksHostAddrType, SocksAuthMethod, SocksPassword, SocksVer, SocksUserName and SocksPort properties and the Connect method.

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com

result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## SocksPort

---

### Summary:

Allows you to select a SOCKS server port other than the well-known port to connect through the SOCKS server.

### Description:

This property must contain the port number of the port of socks server through which the connection is to be made. This should be set before the connection is made through the socks server.

### Type:

Integer

### Default Value:

The default value of this property is 1080.

### See Also:

SocksHostAddrType, SocksAuthMethod, SocksPassword, SocksVer, SocksUserName and SocksServer properties and the Connect method.

### Example:

```
Telnet.SocksHostAddrType = SOCKS_ADDR_DNS
Telnet.SocksAuthMethods = "2"
Telnet.SocksUsername = "joe"
Telnet.SocksPassword = "jim007"
Telnet.SocksVer = SOCKS_VERSION5
Telnet.SocksServer = socksserver.mydomain.com
Telnet.SocksPort = 8080
result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
if (result == 0)
    alert ("Connected");
else
    alert ("Error: " + result );
}.
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## Port

---

### Summary:

The Port property specifies the port on the remote server on which the Telnet service resides.

### Description:

This is the port on the remote server to which a Telnet connection will be made. For most systems, Telnet services listen for a connection request on port 23. However, other ports may be assigned to this.

If the application will be connecting to Telnet services on a different port, then the Port property must be set before the connection is made. An application may even want to query the user for the correct port before connecting. The ActiveX control does not verify the setting of the Port property and will accept any value.

### Type:

Integer

### Default Value:

The default value of this property is 23.

### See Also:

Connect () method.

### Example:

```
//this connects to the server and automatically logs in the user.  
  
//connect and log in automatically  
Telnet1.LoginPrompt = "Welcome: ";  
Telnet1.PasswordPrompt = "Enter password: ";  
Telnet1.CommandPrompt = "#";  
Telnet1.Port = 2000;  
// connect  
result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")  
if (result == 0)  
    alert ("Connected");  
else  
    alert ("Error: " + result );
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## ReceiveData

---

### Summary:

This property contains the data that is sent by the server.

### Description:

The ReceivedData property contains the data that has been sent by the Telnet server. This property is set by the Receive method. The ReceiveCount property can be used to find out the number of bytes of data waiting in the receive buffer before calling the Receive method. Once the Receive method has been called, the application can use ReceivedData property to read the server reply. This property can only be read while a connection is established.

### Type:

String

### Default Value:

There is no default value for this property.

### Note:

This property is read only.

### See Also:

Receive Count property and Receive method.

### Example:

```
// This function reads the data received from the server.
Function srvThread()
{
    // check if there is anything to read
    if (Telnet1.ReceiveCount>0)
    {
        //get the text
        Telnet1.Receive();

        //check if the server response will be received in the buffer or file.
        If (Telnet1.ReceiveInFile != true)

        //and display it if it is in the buffer
        sdlg.value+=Telnet1.ReceivedData();
    }
    // keep the thread running
    sthread=window.setTimeout("srvThread()",100);
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## ReceiveCount

---

### Summary

Specifies the number of bytes of data in the receive buffer.

### Description

The ReceiveCount property specifies how many bytes of data is available in the receive buffer. This property is read-only and can only be accessed while a session is established. After sending a command or while waiting for the prompts, application should check the value of this property and call the Receive method and ReceivedData property to read the data sent by the server. Because of the interrupt driven nature of network communications, the value of the ReceiveCount property can change quickly at any time. An application should not rely on this value for an extended period of time without rechecking it.

### Type:

Integer

### See Also:

ReceivedData property and Receive method.

### Example:

```
// This function reads the data received from the server.
Function srvThread()
{
    // check if there is anything to read
    if (Telnet1.ReceiveCount>0)
    {
        //get the text
        Telnet1.Receive();

        //check if the server response will be received in the buffer or file.
        If (Telnet1.ReceiveInFile != true)

        //and display it if it is in the buffer
        sdlg.value+=Telnet1.ReceivedData();
    }
    // keep the thread running
    sthread=window.setTimeout("srvThread()",100);
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## ReceiveInFile

---

### Summary:

Tells the Object to store all the received data in a file.

### Description:

If ReceiveInFile is set to true, all the received data is stored in a file and the file name is returned in ReceivedDataFileName property. This is necessary when receiving binary data. This property can be set before or after the connection is made.

### Type:

Boolean

### Default Value:

The default value is FALSE.

### See Also:

Binary, ReceivedDataFileName and ReceivedDataPath properties.

### Example:

```
//this connects to the server and automatically logs in the user.
Function autoconnect()
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";
    Telnet1.ReceiveInFile = true;
    Telnet1.ReceivedDataPath = "c:\temp";
    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
    sthread=window.setTimeout("SrvThread()",100);
}

Function srvthread()
{
    //Check if there is anything to read
    if (Telnet1.ReceiveCount>0)
    {
        //Get the data
        Telnet1.Receive()
    }
    //Keep the thread running
    (sthread=window.setTimeout("SrvThread()",100);
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## ReceivedDataPath

---

### Summary:

Sets the path in which the file containing the received data is created.

### Description:

This property is used to set the path of the file in which the received data is stored. If this property is not set, the file will be created in the same folder as the current folder for the process that loaded the object.

### Type:

String

### Default Value:

The default value for this property is the current location of the process that called the Telnet object..

### Note:

ReceiveInFile property must be set to true for this property to have effect.

### Warning:

This property should be set to a path where the user has write permissions, in all cases where users do not have permissions to write to certain folders.

### See Also:

ReceiveInFile property.

### Example:

```
//this connects to the server and automatically logs in the user.
Function autoconnect()
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";
    Telnet1.ReceiveInFile = true;
    Telnet1.ReceivedDataPath = "c:\temp";
    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword","tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
    sthread=window.setTimeout("SrvThread()",100);
}

Function srvthread()
{
    //Check if there is anything to read
    if (Telnet1.ReceiveCount>0)
    {
        //Get the data
```

Distinct Telnet COM Object

```
        Telnet1.Receive()  
    }  
    //Keep the thread running  
    (sthread=window.setTimeout("SrvThread()",100);  
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## ReceivedDataFileName

---

### Summary:

This property contains the name of the file that contains the received data.

### Description:

If the ReceiveInFile property is set to true, this property will be populated by the control with the name of the file where all the received data is saved. The file will be created when the ReceiveInFile property is set to true and deleted after the connection is closed. Every time the application sends something out, the file will be truncated to 0 length so that the new response and old responses do not get mixed up. The same file name will be used to receive all the responses for a single connection.

### Type:

String

### Default Value:

There is no default value. This property is read only.

### See Also:

Binary, ReceiveInFile and ReceivedDataPath properties.

### Example:

```
Function DisplayFileName()  
{  
    alert("Response is saved in file: " + Telnet1.ReceivedDataFileName);  
}
```

[Return to the Telnet Property Summary](#)

[Return to the Telnet Method Summary](#)

---

## Telnet Methods

This section documents all the exported methods of the Telnet object.

### Method Summary

The following lists all the methods and gives a brief description of their function:

Method	Function
<a href="#">Abort</a>	Aborts a telnet session
<a href="#">Connect</a>	Makes a telnet connection to the remote server.
<a href="#">Disconnect</a>	Disconnects the Telnet session.
<a href="#">Receive</a>	Must be called to access the data received.
<a href="#">Send</a>	Sends the buffer or file contents to the server.

---

## Abort ()

---

### Summary:

Aborts the Telnet connection.

### Syntax:

```
Int Abort()
```

### Description:

The Abort method aborts a session to the remote server.

If the connection is successfully reset and closed, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	Command was successful.
6	ERR_TELNET_HANDLE	Internal Telnet error

### See Also:

Connect method.

### Example:

```
Function Cancel()  
{  
    result = Telnet1.Abort();  
    if (result == 0)  
}
```

[Return to the Telnet Method Summary](#)

[Return to the Telnet Property Summary](#)

## Connect ()

### Summary:

Connects to the remote server

### Syntax:

Int Connect (String *Host* , String *User*, String *Password*, String *TerminalType*)

Parameter	Meaning
<i>Host</i>	is a string that contains the host name or IP address (in dotted decimal notation) of the Telnet server to connect to.
<i>User</i>	The name of the user account to be connected.
<i>Password</i>	The password for the user account to be connected.
<i>TerminalType</i>	The terminal type to connect with.

### Description:

The Connect method establishes a connection to the remote server. The host is the name or Internet address (in dotted decimal notation) of the remote server. User and Password parameters are optional. Please check the table provided with the LoginPrompt property to understand the behavior of the control in various cases. Parameters User and Password can be used along with LoginPrompt, PasswordPrompt and CommandPrompt properties to login automatically. The *Terminal Type* must be set to the type of terminal that will be used for the Telnet session. If you are not using any emulation set this to tty. The value "tty" will cause the Telnet server to send no screen control commands. If the application you are writing will, for example, emulate a DEC VT100 terminal, then the *TerminalType* string should be set to "vt100". If a connection is successfully established, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value.

### Note:

If the SocksServer property is non-empty then the connection is made thru the specified socks server.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The connection was successful.
1	ERR_LICENSE	License error
2	ERR_CANNOT_CONNECT	General socket error.
5	ERR_TIMEOUT	A timeout error occurred while connecting or waiting for a prompt.
6	ERR_TELNET_HANDLE	Internal Telnet error
7	ERR_NO_HOST_NAME	The remote telnet host name is not defined.
8	ERR_NO_TERM_TYPE	A terminal type has not been defined.
10	ERR_LOGIN_PROMPT_MISSING	You set the user name but did not specify a login prompt.
11	ERR_PASSWORD_PROMPT_MISSING	You set the password but the password prompt is missing..
14	ERR_FILE_CREATE	The file that is to contain the received data could not be created.

## Distinct Telnet COM Object

15	ERR_INVALID_HOST_NAME	The hostname or IP address is invalid
16	ERR_MAX_CONNECTIONS	You have exceeded the maximum number of concurrent Telnet connection. The default for this is 16 but can be changed through a registry entry.

### See Also:

Disconnect () method and LoginPrompt, PasswordPrompt and CommandPrompt properties.

### Example:

```
//this connects to the server and automatically logs in the user.
{
    //connect and log in automatically
    Telnet1.LoginPrompt = "Welcome: ";
    Telnet1.PasswordPrompt = "Enter password: ";
    Telnet1.CommandPrompt = "#";

    // connect
    result=Telnet1.Connect("MyHost", "MyUser", "MyPassword", "tty")
    if (result == 0)
        alert ("Connected");
    else
        alert ("Error: " + result );
}
```

[Return to the Telnet Method Summary](#)

[Return to the Telnet Property Summary](#)

---

## Disconnect ()

---

### Summary:

Closes the connection to the remote server.

### Syntax:

```
Int Disconnect ()
```

### Description:

Calling the Disconnect method terminates a Telnet session. An application must close all connections it has created before it quits.

The Disconnect method takes no parameters and returns an integer. If a connection is successfully terminated, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The request to disconnect the connection succeeded
6	ERR_TELNET_HANDLE	The attempt to disconnect has failed.

### See Also:

Connect() method.

### Example:

```
Function Disconnect()  
{  
    Telnet1.Disconnect();  
}
```

[Return to the Telnet Method Summary](#)

[Return to the Telnet Property Summary](#)

## Receive ()

This method is used to receive data over a Telnet connection.

### Syntax:

```
Int Receive ()
```

### Description:

The Receive method is called to access the data that has been sent by the Telnet server. Whenever this method is called, as many bytes of data as available are copied to the ReceivedData property or if the ReceiveInFile property is set to true they are stored in a file. To ensure that the method was successfully executed, the application should check the return value.

Before calling this method, the application should first check the value of the ReceiveCount property. If this is non-zero, it should call the Receive method to get the available data. This method can only be called while a connection is established.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The response was successfully received
1	ERR_LICENSE	License error
3	ERR_CONNECT_TO_RECV	You are trying to receive data but you are not currently connected to the Telnet Server.
6	ERR_TELNET_HANDLE	Internal Telnet error.
12	ERR_FILE_OPEN	The file to place the received data in, could not be opened.
13	ERR_FILE_WRITE	The file to place the received data in, could not be written to.

### See Also:

Send () method and ReceiveInFile, ReceivedDataFileName and ReceiveCount properties.

### Example:

```
// This function reads the data received from the server.
Function srvThread()
{
    // check if there is anything to read
    if (Telnet1.ReceiveCount>0)
    {
        //get the text
        Telnet1.Receive();

        //check if the server response will be received in the buffer or file.
        If (Telnet1.ReceiveInFile != true)

        //and display it if it is in the buffer
        sdlg.value+=Telnet1.ReceivedData();
    }
    // keep the thread running
    sthread=window.setTimeout("srvThread()",100);
}
```

[Return to the Telnet Method Summary](#)

Distinct Telnet COM Object

[Return to the Telnet Property Summary](#)

## Send ()

### Summary:

Sends the buffer or file contents to the server

### Syntax:

Int Send (String *Buffer*, Boolean *IsFile*)

Parameter	Meaning
<i>Buffer</i>	Either the string to be sent or the name of a file that contains the data to be sent.
<i>IsFile</i>	When set to TRUE this specifies that <i>Buffer</i> is a filename, when set to false this specifies that <i>Buffer</i> is the string to be sent.

### Description:

The Send method is used to send data to the Telnet server. Care should be taken not to exceed the buffer and transport capabilities of the underlying protocol stack.

Usually, this method is set in response to the user input (for example, entering a command to be executed on the Telnet server). The response to these user commands can be read from the ReceivedData property after a call to the Receive method. The ReceiveCount property can be used to check how much data (if any) is available to be read.

### Note:

If you wish to receive the server response in a file instead of a buffer you must set the ReceiveInFile property to TRUE. In this case the received data will be saved to a file having the name specified by the ReceivedDataFileName property.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The data was successfully sent.
4	ERR_CONNECT_TO_SEND	You must make a telnet connection before you can send any data to the Telnet server.
5	ERR_TIMEOUT	A timeout occurred while trying to send the data.
6	ERR_TELNET_HANDLE	Internal Telnet error.
12	ERR_FILE_OPEN	The file containing the data to be sent could not be opened.
14	ERR_FILE_CREATE	The file that is to contain the data received in response to the data being sent could not be created. Check that you have the correct permissions to create files under the path you have specified in the ReceivedDataPath property.

### See Also:

Receive () method and ReceiveInFile, ReceivedDataFileName and ReceiveCount properties.

### Example:

```
// this JavaScript function sends a command to the server

function Send()
{
    // send the command
}
```

## Distinct Telnet COM Object

```
result=Telnet1.Send ("ls -l" + \r\n", false)
if (result == 0)
    alert ("Command sent successfully");
else
    alert ("Error: " + result );
}
```

[Return to the Telnet Method Summary](#)

[Return to the Telnet Property Summary](#)



---

## 4 – Remote Copy Object

### Remote Copy Overview

Remote Copy (RCP) allows the copying of files between local and remote systems or between two remote systems.

### What needs to be set up on the remote system

In order to make a connection using Remote Copy (RCP) you must register the PC running your application in the hosts.equiv file. Here is an example of how to do this:

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 RCP 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 you need to run RCP between two UNIX systems, you will need to do all of the above plus:

When running the RCP 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.

### How to Use the RCP COM Object

To use the RCP COM Object in your ASP or HTML page you must embed it on the page.

## How to embed the control in a HTML page

In HTML you create the RCP Object as follows:

```
<OBJECT ID="Rcp1" WIDTH="1" HEIGHT="1"  
  CLASSID="CLSID:5E936CD4-AE65-4175-8DBF-2B962E0DFC1B"  
  CODEBASE="./dsv_rcp.cab">  
</OBJECT>
```

Note that you need to create a cab file that contains the dsv\_rcp.ocx and dsv\_rlib.dll files as shown in the sample that comes with this product.

## How to embed the control in an ASP page

In ASP you create the RCP Object by adding this line.

```
Set RCPObj = Server.CreateObject ("DistinctServerRCP.RCPCtrl")
```

## Registry Entries

The following registry entries may be used when using this object on HTML pages. The entries must be made under:

HKLM\Software\Distinct\Dlls\Web\Rlib

RcpTimeout REG\_DWORD 1-999

Specifies the 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 error buffer contains all the error messages sent by the server when a file copy fails. When the value specified here is reached, the error messages returned will be truncated. 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 method will terminate. The default value is 50. The default value may need to be increased when large directories are being copied.

---

## RCP Properties

The following is a description of all the properties that can be set in the RCP COM Object.

### Property Summary

The following lists all the properties of the RCP COM Object and gives a brief summary of their function:

Property	Function
<a href="#">DestFile</a>	Specifies the destination filename or directory.
<a href="#">DestHost</a>	Specifies the name or IP address of the remote system.
<a href="#">DestUser</a>	Specifies the User Account name on the remote system.
<a href="#">ErrorMessage</a>	Contains an error response from the remote system.
<a href="#">File Attributes</a>	Used to read the file attributes of files being transferred.
<a href="#">FileDate</a>	Specifies the date of the last modification of the file or directory in FileName.
<a href="#">FileName</a>	Specifies the name of the file or directory that is currently being copied.
<a href="#">FileTime</a>	Specifies the time of the last modification of the file or directory specified in FileName
<a href="#">FileSize</a>	Specifies the size of the file or directory specified in FileName.
<a href="#">FileType</a>	Specifies whether the contents of FileName is a filename or directory
<a href="#">Options</a>	Specifies the copy options to be used.
<a href="#">SrcFile</a>	Specifies the source file or directory name.
<a href="#">SrcHost</a>	Specifies the name or IP address of the source system.
<a href="#">SrcUser</a>	Specifies the source user name.

---

## DestFile

---

### Summary

This property specifies the destination file name or folder.

### Description:

The DestFile property specifies the name of the file or folder to be copied. The destination could be a remote host or the local machine. If you are copying a file from a remote system to the local system, then both the DestHost and DestUser properties must be set to an empty string and DestFile should contain the absolute path of the file on the local system. If the destination is a remote host, DestFile can contain a filename or a complete path.

This property must be set before calling the FileCopy () method. The DestFile property can contain either a file or a folder name. If it is set to a folder name, it should already exist on the destination.

### Type:

String

### Default Value:

There is no default value for this property. It must be set prior to calling FileCopy().

### Note:

When copying more than one file in a single operation this property must be set to a folder name that already exists. If the filename or folder name contains spaces, it must be enclosed in double quotes.

### See Also:

DestHost and DestUser properties and FileCopy () method.

### Examples:

#### To copy a file:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "folder1/abc.txt";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
Rcp1.DestFile = "c:\temp\abc.txt";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

#### To copy a folder:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
```

## Distinct RCP COM Object

```
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "folder1";
Rcp1.Options = OPTION_RECURSIVE;

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
Rcp1.DestFile = "temp";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

### To copy a file and a folder:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
// option has to be set to recursive
Rcp1.Options = OPTION_RECURSIVE;
Rcp1.SrcFile = "folder1/abc.txt folder2";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
// destination has to be an existing folder
Rcp1.DestFile = "folder1";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

### To copy all the files from a folder:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "folder1/*.*";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
// destination has to be an existing folder
Rcp1.DestFile = "c:\temp\folder1";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
```

Distinct RCP COM Object

```
else  
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## DestHost

---

### Summary

This property specifies the name or IP address of the destination system.

### Description:

The DestHost property specifies the name or Internet address of the destination system. This property must be set before calling FileCopy().

If the destination is the local system, then this property must be set to an empty string.

If the destination is a remote system, then there are three possible ways of specifying the destination.

1. By specifying the **Machine Name**. An application only needs to specify the name of the remote system if the system is located on the same network as the local PC or if its IP address is defined in the local host table. If the remote server is not on the local network, then the underlying protocol will route the traffic through a gateway. If the remote server is not defined in the local host table, then the underlying protocol will contact the domain server to resolve the IP address of the remote system.
2. By specifying the **Machine and Domain Name**. An application needs to specify the machine name and the domain name if the remote system is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *speedy.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the domain server to resolve the IP address of the remote system.
3. By specifying the **IP Address of the remote system**. Sometimes the user knows only the IP address of the remote server that he or she wants to use. In this case, the IP address can be entered in what is known as the dotted decimal notation (for example, *127.43.101.12*). If the remote server identified by this address is not on the local network, then the underlying protocol will route the traffic through a gateway.

### Type:

String

### See Also:

DestFile and DestUser properties and FileCopy() method.

### Example:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "c:\temp\abc.txt";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
Rcp1.DestFile = "folder1/abc.txt";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
```

Distinct RCP COM Object

else

alert ("Error: " + result + "\r\n" + Rcp1.**ErrorMessage**);

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## DestUser

---

### Summary

This property specifies the name of the user account to which the file or files will be copied.

### Description:

The DestUser property specifies the user name to be used to login to the remote system. The destination can be a remote host or the local system. If the destination is the local machine, then both the DestHost and DestUser properties must be set to an empty string.

If the destination is a remote host, then the DestUser property must be set to the login name of the user in whose account the files will be copied.

This property must be set before FileCopy () is called.

### Type:

String

### See Also:

DestHost and DestFile properties and FileCopy () method.

### Example:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "c:\temp\abc.txt";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
Rcp1.DestFile = "folder1/abc.txt";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## ErrorMessage

---

### Summary:

This property contains the remote host response when the connection could not be established or a copy operation fails

### Description:

The ErrorMessage property contains an error message if the connection could not be established. If the copy operation could not be properly executed for any reason, then it contains the error messages sent by the server. This property will contain the exact response from the server explaining the error that occurred.

### Type:

String

### Note:

This property is read only.

### See Also:

FileCopy () method

### Example:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "folder1/abc.txt";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
Rcp1.DestFile = "c:\temp\abc.txt";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## FileAttributes

---

### Summary

Contains the file attributes of the files being transferred.

### Description:

The FileAttributes property can be referenced during a directory transfer to see the attributes of files being transferred.

The value of the FileAttributes property can be any combination of the values mentioned below.

Value	Assigned to:	Meaning
1	ATTRIBUTE_EXECUTE	User has execute permission
2	ATTRIBUTE_WRITE	User has write permissions
4	ATTRIBUTE_READ	User has read permissions

### Type:

Short

### Note:

This property is read only.

### See Also:

FileCopy() method

### Example:

This Javascript shows how when transferring a folder you can create a timer to update a text box, and display the file attributes of the file currently being transferred:

```
Create a timer like this:  
sthrad=window.setTimeout("srvThread()",100);  
  
// this function will be called every 100 ms  
function srvThread()  
{  
    // create a string with the file attributes  
    rstr=new String(Rcp1.FileName + "" + Rcp1.FileSize + "" + Rcp1.FileDate + "" +  
Rcp1.FileTime + "" + Rcp1.FileAttributes)  
    //and display it  
    sdlg.value=rstr;  
    sthrad=window.setTimeout("srvThread()",100);  
}
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## FileDate

---

### Summary

This property specifies the date of the last modification of the file or folder being copied.

### Description:

The FileDate property specifies the date of the last modification of the file or folder specified in the FileName property. This property is only set when the Options property is set to OPTION\_PRESERVE or OPTION\_BOTH.

This property can be referenced during a directory transfer to see the date of files that are being transferred.

### Type:

String

### Note:

This property is read only.

### See Also:

Options property.

### Example:

This Javascript shows how when transferring a folder you can create a timer to update a text box, and display the file attributes of the file currently being transferred:

```
Create a timer like this:  
sthrad=window.setTimeout("srvThread()",100);  
  
// this function will be called every 100 ms  
function srvThread()  
{  
    // create a string with the file attributes  
    rstr=new String(Rcp1.FileName + " " + Rcp1.FileSize + " " + Rcp1.FileDate + " " +  
Rcp1.FileTime + " " + Rcp1.FileAttributes)  
    //and display it  
    sdlg.value=rstr;  
    sthrad=window.setTimeout("srvThread()",100);  
}
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## FileName

---

### Summary

Specifies the name of the file or directory that is currently being copied.

### Description:

The FileName property specifies the name of the file or directory that is currently being copied. This property can be used during a directory transfer to display the name of the files that are being transferred.

### Type:

String

### Note:

This property is read only.

### See Also:

FileTime, FileSize and FileType properties and FileCopy () method.

### Example:

This Javascript shows how when transferring a folder you can create a timer to update a text box, and display the file attributes of the file currently being transferred:

```
Create a timer like this:  
sthrad=window.setTimeout("srvThread()",100);  
  
// this function will be called every 100 ms  
function srvThread()  
{  
    // create a string with the file attributes  
    rstr=new String(Rcp1.FileName + "" + Rcp1.FileSize + "" + Rcp1.FileDate + "" +  
Rcp1.FileTime + "" + Rcp1.FileAttributes)  
    //and display it  
    sdlg.value=rstr;  
    sthrad=window.setTimeout("srvThread()",100);  
}
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## FileSize

---

### Summary

This property contains the size of the File or folder having the name that is currently in the FileName property.

### Description:

The FileSize property specifies the size of the file or directory specified by the FileName property. This property can be referenced during a file or directory transfer to display the size of the file(s) that are being transferred.

The FileType property tells the application whether the FileName property specifies a file or a directory. If the FileType is FILETYPE\_DIRECTORY, then the value of the FileSize property is 0.

### Type:

Integer

### Note:

This property is read only.

### See Also:

FileType, FileName and FileTime properties and FileCopy () method.

### Example:

This Javascript shows how when transferring a folder you can create a timer to update a text box, and display the file attributes of the file currently being transferred:

```
Create a timer like this:  
sthrad=window.setTimeout("srvThread()",100);  
  
// this function will be called every 100 ms  
function srvThread()  
{  
    // create a string with the file attributes  
    rstr=new String(Rcp1.FileName + " " + Rcp1.FileSize + " " + Rcp1.FileDate + " " +  
Rcp1.FileTime + " " + Rcp1.FileAttributes)  
    //and display it  
    sdlg.value=rstr;  
    sthrad=window.setTimeout("srvThread()",100);  
}
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## FileTime

---

### Summary

This property specifies the time that the file or folder specified in the FileName property was last modified.

### Description:

The FileTime property specifies the time of the last modification of the file or directory specified in the FileName property. This property is set to an empty string unless the Options property is set to OPTION\_PRESERVE or OPTION\_BOTH.

This property can be referenced during a file or directory transfer to display the time on the file that is being transferred.

### Type:

String

### Note:

This property is read only.

### See Also:

FileName, FileSize, FileType properties and FileCopy () method.

### Example:

This Javascript shows how when transferring a folder you can create a timer to update a text box, and display the file attributes of the file currently being transferred:

```
Create a timer like this:  
sthrad=window.setTimeout("srvThread()",100);  
  
// this function will be called every 100 ms  
function srvThread()  
{  
    // create a string with the file attributes  
    rstr=new String(Rcp1.FileName + " " + Rcp1.FileSize + " " + Rcp1.FileDate + " " +  
Rcp1.FileTime + " " + Rcp1.FileAttributes)  
    //and display it  
    sdlg.value=rstr;  
    sthrad=window.setTimeout("srvThread()",100);  
}
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## FileType

---

### Summary

This property indicates whether the name specified in the FileName property refers to a file or folder.

### Description:

The FileType property indicates whether the name specified in the FileName property refers to a file or a folder. The value of the FileType property can be one of the following values:

Value	Assigned to:	Meaning
0	FILETYPE_DIRECTORY	Folder
1	FILETYPE_FILE	File

This property can be referenced during a file or directory transfer to display the type of the file.

### Type:

Short

### Note:

This property is read only.

### See Also:

FileName, FileSize, FileTime properties and FileCopy () method.

### Example:

```
Create a timer like this:  
sthread=window.setTimeout("srvThread()",100);  
  
// this function will be called every 100 ms  
function srvThread()  
{  
    // create a string with the file attributes  
    rstr=new String(Rcp1.FileName + "" + Rcp1.FileSize + "" + Rcp1.FileDate + "" +  
Rcp1.FileType + "" + Rcp1.FileAttributes)  
    //and display it  
    sdlg.value=rstr;  
    sthread=window.setTimeout("srvThread()",100);  
}
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

## Options

### Summary

Specifies any required copy options.

### Description:

The Options property is used to specify the various options during the copy operation. The value of the Options property can be one of following:

Value	Assigned to:	Meaning
0	OPTION_NONE	No option
1	OPTION_PRESERVE	Preserves file date and time
2	OPTION_RECURSIVE	Recursive copy
3	OPTION_BOTH	Preserves the date and time and make a recursive copy.

If the Options property is set to OPTION\_PRESERVE, then the date and time stamp of the source file will be copied to the destination file. Otherwise, the destination file will not have the same date and time stamp as the source file.

If the Options property is set to OPTION\_RECURSIVE, then the RCP Object will not only copy the files in the source directory, but also in all of its subdirectories. This applies only if the source is a directory.

To specify both options set the Options property to OPTION\_BOTH. To transfer a directory, this property should be set to OPTION\_RECURSIVE or OPTION\_BOTH.

### Type:

Short

### Default Value:

The default value for this property is OPTION\_NONE.

### See Also:

FileCopy method

### Example:

```
// assuming we want to preserve date and time of the files and transfer recursively
Rcp1.Options = OPTION_BOTH;

// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";

// we want to transfer folder named folder1
Rcp1.SrcFile = "folder1";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";

// since the SrcFile is a folder name, DestFile has to be a folder name
Rcp1.DestFile = "c:\temp\folder1";
```

## Distinct RCP COM Object

```
//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## SrcFile

---

### Summary

This property specifies the source file or folder name.

### Description:

The SrcFile property specifies the source file or folder name. The source can be a remote host or the local machine. If the source is the local machine, then both the SrcHost and SrcUser properties must be set to an empty string and SrcFile should contain the absolute path.

This property must be set before a copy operation is invoked. The SrcFile property may contain one or more file or folder names. Multiple file or folder names must be separated by spaces. If SrcFile contains multiple names then DestFile should be a directory.

### Type:

String

### Note:

File names with spaces must be placed in double quotes.

### See Also:

SrcHost and SrcUser properties

### Example:

```
// assuming we want to preserve date and time of the files and transfer recursively
Rcp1.Options = OPTION_BOTH;

// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";

// we want to transfer folder named folder1
Rcp1.SrcFile = "folder1";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";

// since the SrcFile is a folder name, DestFile has to be a folder name
Rcp1.DestFile = "c:\temp\folder1";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## SrcHost

---

### Summary

This property specifies the name or IP address of the system from which a file will be copied.

### Description:

The SrcHost property specifies the name or IP address of the system from which a file will be copied when the file is on the remote system. This property must be set before a copy operation.

If the source is the local system, then the SrcHost property must be set to an empty string.

If the source is a remote system, then the system must be specified in one of the following ways:

#### Machine Name

You can specify the Machine name if the IP address is defined in the local host table or the system is on the same network as that running the Distinct RCP Object.

#### Machine and Domain Name

An application needs to specify the machine name and the domain name if the remote server is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *speedy.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the domain server to resolve the IP address of the server.

#### Internet Address

The IP address of the system in dotted decimal notation (for example, *127.43.101.12*).

### Type:

String

### See Also:

SrcUser and SrcFile properties

### Example:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "folder1/abc.txt";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";
Rcp1.DestFile = "c:\temp\abc.txt";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## SrcUser

---

### Summary

This property specifies the name of the account from where the file will be copied.

### Description:

The SrcUser property specifies the user account name for the source system. The source system could be a remote host or the local system. If the source is the local system, then both the SrcHost and SrcUser properties must be set to an empty string.

If the source is a remote system, then the SrcUser property must be set to a user account name that will be used to login and copy the file. This property must be set before a copy operation is invoked.

### Type:

String

### Note:

The user logging in must have correct permissions to copy the file to the specified folder if it is not in his or her own home folder.

### See Also:

SrcHost and SrcUser properties.

### Example:

```
// assuming we want to preserve date and time of the files and transfer recursively
Rcp1.Options = OPTION_BOTH;

// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";

// we want to transfer folder named folder1
Rcp1.SrcFile = "folder1";

// assuming that the destination host is local
Rcp1.DestHost = "";
Rcp1.DestUser = "";

// since the SrcFile is a folder name, DestFile has to be a folder name
Rcp1.DestFile = "c:\temp\folder1";

//do copying
var result = Rcp1.FileCopy ();
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Property Summary](#)

[Return to RCP Method Summary](#)

---

## RCP Methods

This section documents all the exported methods of the RCP object

### Method Summary

The following lists all the methods and gives a brief description of their function:

Method	Function
<a href="#">Abort</a>	Aborts the current action.
<a href="#">FileCopy</a>	Copies a file or folder to or from a remote system.

---

## Abort ()

---

### Summary:

Aborts the FileCopy operation.

### Syntax:

```
Int Abort()
```

### Description:

The Abort method allows you to abort a copy operation in the case where multiple files or folders are being copied. When this method is called the file currently copied will complete but the subsequent files to be copied will be aborted.

The Abort method takes no parameters and returns 0.

### Warning:

This method can only abort the copying of a file before the actual copying is started. Abort may not be used if the file copy is between two remote systems.

### See Also:

FileCopy () method.

[Return to RCP Method Summary](#)

[Return to RCP Property Summary](#)

## FileCopy

### Summary:

Copies the files specified from one system to another

### Syntax:

```
Int FileCopy ()
```

### Description:

The FileCopy() method must be called to transfer files from one system to the other. Before calling the FileCopy method, the SrcHost, SrcUser, DestHost, DestUser, DestFile and SrcFile properties must be set. If the transfer is from remote system to the local system, SrcUser must also be set. If the file(s) can be successfully copied, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value.

In a folder transfer, the content of the source folder is transferred to the specified destination folder. If the destination folder does not exist, only one level of directory is created on the server side. In a file transfer, the destination folder must already exist otherwise the transfer will fail.

### Warning:

For the local system, you need to specify the absolute path of the file or folder otherwise the copy operation will fail.

### Return Values:

Value	Assigned to	Meaning
0	ERR_SUCCESS	The copy was successful
1	ERR_CANNOT_COPY	Unable to copy file. The problem is most likely due to insufficient memory.
2	ERR_COPY_ABORTED	The copy operation has been aborted.
4	ERR_INVALID_OPTION	The specified option is out of range.
5	ERR_LICENSE	License error detected.
6	ERR_HOST_NOT_DEFINED	At least one host has not been specified.
7	ERR_FILES_NOT_DEFINED	The file names have not been defined.
8	ERR_RCP_FAILED	A socket error has occurred.

### See Also:

SrcHost, DestHost, SrcFile and SrcUser properties

### Example:

```
// assume that the source host is remote
Rcp1.SrcHost = "110.234.43.1";
Rcp1.SrcUser = "me";
Rcp1.SrcFile = "folder1/abc.txt";

// assuming that the destination host is remote
Rcp1.DestHost = "pineapple.distinct.com";
Rcp1.DestUser = "Sam";
Rcp1.DestFile = "abc.txt";

//do copying
var result = Rcp1.FileCopy ();
```

## Distinct RCP COM Object

```
if (result == 0)
    alert ("File copied successfully");
else
    alert ("Error: " + result + "\r\n" + Rcp1.ErrorMessage);
```

[Return to RCP Method Summary](#)

[Return to RCP Property Summary](#)



---

## 5 – Remote Commands Object

### Remote Commands Overview

The Remote Command execution commands allow a process on a host to cause a program to be executed on a remote system. There are two commands for this, RSH and REXEC. The essential difference between these two is the method that is used to verify that the user invoking the command has adequate permission to execute a particular process on the remote system. To use RSH the user has to have super user privileges so that a reserved port can be bound on the local host. RSH does not require the user to enter the account password, since the rsh server will check that the user name does have the appropriate privileges to be executing commands. Rexec on the other hand requires a user name and password, which will traverse the wire in clear text. Rexec does not require that the user have super user privileges.

### You must be a Trusted Host to use RSH

In order to make a connection using RSH 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. Note that different rshd may have slightly different security requirements. Check your server documentation for exact information on how to set this up on your particular server.

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 RSH successfully:

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

```
unix1.mydomain.com:  
192.0.0.3 mypc mypc.mydomain.com
```
2. Add the following entries in the `/etc/hosts.equiv` file:  

```
mypc  
mypc.mydomain.com
```
3. You must also add the host to the `/.rshosts` for superuser access. Otherwise it needs to be added to the `rhosts` directory in the user's home directory.

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.

You must also add the host to the `/.rshosts`

## How to Use the Remote Commands Object

The Remote Commands Object must be added to your page.

### How to embed the control in a HTML page:

To add the Remote Commands Object to an HTML page add this tag.

```
<OBJECT ID="Rcmd1" WIDTH="1" HEIGHT="1"  
  CLASSID="CLSID:5E1A5502-5D05-44DC-A2F3-395792AF7BC3">  
  CODEBASE="./dsv_rcmd.cab">  
</OBJECT>
```

Note that you need to create a cab file that contains the dsv\_rcmd.ocx and dsv-rlib.dll files as shown in the sample that comes with this product.

### How to embed the control in an ASP page:

To create the Remote Commands Object on an ASP page add the following line:

```
Set RcmdObj = Server.CreateObject ("DistinctServerRCMD.RCMDCtrl")
```

## Registry Entries

The following are the registry entries that can be made to change the behavior of this object. To make any of these entries you must first create the following key:

```
HKLM\SOFTWARE\Distinct\DLLS\Web\RLIB
```

### REXEC registry entries

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.

### RSH registry entries:

RshTimeout REG\_DWORD 1-999

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

## Distinct Remote Commands COM Object

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.

---

## Remote Commands Properties

The following is a description of all the properties that can be set in the Remote Commands COM Object.

### Property Summary

The following lists all the properties of the Remote Commands COM Object and gives a brief summary of their function:

Property	Function
<a href="#">CommandCompleted</a>	Specifies whether the rexec or rsh command execution is completed.
<a href="#">ErrorMessage</a>	Contains the server response in the case of an error.
<a href="#">ReceivedData</a>	Contains the data sent by the server.
<a href="#">ReceivedDataFileName</a>	Contains the name of the file where all the received data is saved
<a href="#">ReceivedDataPath</a>	Contains the path for the file in ReceivedDataFileName.
<a href="#">ReceiveInFile</a>	Specifies whether the server response should be saved in a file.

---

## CommandCompleted

---

### Summary

Specifies whether the rexec or rsh command has completed.

### Description:

After the rexec and rsh commands are executed, an application can check CommandCompleted property to see if the execution of the command is completed. Once the client has been informed by the server that the server has closed the connection (after all the responses are sent), this property is set to TRUE.

### Type:

Boolean

### Default Value:

This property is read-only and has no default value.

### See Also:

Rexec () and Rsh () methods.

### Example:

```
// this function displays the result of rexec or rsh commands function
Function GetData()
{
    if (!Rcmd1.ReceiveInFile)
    {
        //showing the server reply
        srply.value="Server returned:\r\n"+Rcmd1.ReceivedData+"\r\n";

        // check if the server reply is completed
        if (Rcmd1.CommandCompleted) return;
        Rcmd1.Receive ();
        //if the reply received is not complete, continue waiting
        window.setTimeout("ShowResult()",200);
    }
    else
    {
        //if the reply received is not complete, continue waiting
        if (!Rcmd1.CommandCompleted)
        {
            Rcmd1.Receive ();
            window.setTimeout("ShowResult()",200);
            return;
        }

        //show the filename server reply was saved into
        srply.value="Result saved in file: " + Rcmd1.ReceivedDataFileName + "\r\n";
    }
}
```

[Return to Remote Commands Property Summary](#)

Distinct Remote Commands COM Object

[Return to Remote Commands Method Summary](#)

---

## ErrorMessage

---

### Summary:

Contains the server response in case of an error.

### Description:

The ErrorMessage property contains an error message if a connection could not be established. If the command could not be properly executed for some other reason, then it contains the message sent by the server. This property will contain the exact response from the server explaining the error that occurred.

### Type:

String

### Default Value:

There is no default value for this method.

### See Also:

Rexec () and Rsh () methods

### Example:

```
//this function performs rexec
function rexec()
{
    //determining the "save to file" option
    if (tofile.checked)
        Rcmd1.ReceiveInFile=true;
    else
        Rcmd1.ReceiveInFile=false;

    //calling rexec
    var result=Rcmd1.Rexec("HostName", "User", "Password", "Command");
    if (result == 0)
    {
        //starting server monitoring thread to display result
        alert ("Command executed successfully");
    }
    else
        alert ("Error: " + result + "\r\n" + Rcmd1.ErrorMessage);
}
```

[Return to Remote Commands Property Summary](#)

[Return to Remote Commands Method Summary](#)

---

## ReceivedData

---

### Summary

This property contains the data sent by the server.

### Description

The ReceivedData property is used to access data that has been sent by the server. This property is set in response to Rexec and Rsh methods.

### Type:

String

### Note:

This property is not populated if the ReceiveInFile property is set to TRUE.

### Default Value:

There is no default value for this property

### See Also:

ReceiveInFile property and Rexec () and Rsh () methods

### Example:

```
// this function displays the result of rexec or rsh commands function ShowResult()
Function GetData ()
{
    if (!Rcmd1.ReceiveInFile)
    {
        //showing the server reply
        srply.value="Server returned:\r\n"+Rcmd1.ReceivedData+"\r\n";

        // check if the server reply is completed
        if (Rcmd1.CommandCompleted) return;
        Rcmd1.Receive ();
        //if the reply received is not complete, continue waiting
        window.setTimeout("ShowResult()",200);
    }
    else
    {
        //if the reply received is not complete, continue waiting
        if (!Rcmd1.CommandCompleted)
        {
            Rcmd1.Receive ();
            window.setTimeout("ShowResult()",200);
            return;
        }

        //show the filename server reply was saved into
        srply.value="Result saved in file: " + Rcmd1.ReceivedDataFileName + "\r\n";
    }
}
```

[Return to Remote Commands Property Summary](#)

Distinct Remote Commands COM Object

[Return to Remote Commands Method Summary](#)

---

## ReceivedDataFileName

---

### Summary:

Contains the name of the file that contains the data sent by the server.

### Description:

If ReceiveInFile property is set to true, this property will contain the name of the file where all the received data is saved. The file will be created when the ReceiveInFile property is set to true. The main purpose for this is to be able to receive binary data. Every time a command is executed, the file will be truncated to 0 length so that the new response and old responses do not get mixed up. The same file name will be used to receive all the responses for a single command.

### Type:

String

### Default Value:

This property has no default value.

### See Also:

ReceiveInFile and ReceivedDataPath properties.

### Example:

```
// this function displays the result of rexec or rsh commands function ShowResult()
Function GetData()
{
    if (!Rcmd1.ReceiveInFile)
    {
        //showing the server reply
        srply.value="Server returned:\r\n"+Rcmd1.ReceivedData+"\r\n";

        // check if the server reply is completed
        if (Rcmd1.CommandCompleted) return;
        Rcmd1.Receive ();
        //if the reply received is not complete, continue waiting
        window.setTimeout("ShowResult()",200);
    }
    else
    {
        //if the reply received is not complete, continue waiting
        if (!Rcmd1.CommandCompleted)
        {
            Rcmd1.Receive ();
            window.setTimeout("ShowResult()",200);
            return;
        }

        //show the filename server reply was saved into
        srply.value="Result saved in file: " + Rcmd1.ReceivedDataFileName + "\r\n";
    }
}
```

[Return to Remote Commands Property Summary](#)

Distinct Remote Commands COM Object

[Return to Remote Commands Method Summary](#)

---

## ReceivedDataPath

---

### Summary:

This property is used to set the path where the file containing the server responses will be stored.

### Description:

This property is used to set the path of the file in which the received data is stored if ReceiveInFile property is set to true. The path must be set before the Rsh or Rexec method is called.

### Type:

String

### Warning:

Make sure that the path selected has write permissions for any user, otherwise the method will fail.

### Default Value:

If this property is not set the default path will be the current folder for the process that loaded the object.

### See Also:

ReceiveInFile and ReceivedDataFilename properties.

### Example:

```
Rcmd1.ReceivedDataPath = "c:\userarea\temp"
```

[Return to Remote Commands Property Summary](#)

[Return to Remote Commands Method Summary](#)

---

## ReceiveInFile

---

### Summary:

Specifies whether all the server responses will be saved to a file or to the ReceivedData property.

### Description:

If this is set to true, all the received data is stored in a file and the file name is returned in ReceivedDataFileName property. This property is useful in receiving the binary data.

### Type:

Boolean

### Default Value:

The default value of this property is FALSE which means that the data will be stored in the ReceivedData property.

### See Also:

ReceivedDataPath and ReceivedDataFileName properties.

### Example:

```
// this function displays the result of rexec or rsh commands function ShowResult()
Function GetData()
{
    if (!Rcmd1.ReceiveInFile)
    {
        //showing the server reply
        srply.value="Server returned:\r\n"+Rcmd1.ReceivedData+"\r\n";

        // check if the server reply is completed
        if (Rcmd1.CommandCompleted) return;
        Rcmd1.Receive ();
        //if the reply received is not complete, continue waiting
        window.setTimeout("ShowResult()",200);
    }
    else
    {
        //if the reply received is not complete, continue waiting
        if (!Rcmd1.CommandCompleted)
        {
            Rcmd1.Receive ();
            window.setTimeout("ShowResult()",200);
            return;
        }

        //show the filename server reply was saved into
        srply.value="Result saved in file: " + Rcmd1.ReceivedDataFileName + "\r\n";
    }
}
```

[Return to Remote Commands Property Summary](#)

[Return to Remote Commands Method Summary](#)

---

## Remote Objects Methods

This section documents all the exported methods of the Remote Objects COM object

### Method Summary

The following lists all the methods and gives a brief description of their function:

Method	Function
<a href="#">Abort</a>	Aborts the Rexec or Rsh method
<a href="#">Receive</a>	Places the received data in a property or file.
<a href="#">Rexec</a>	Executes a specified command on a remote system.
<a href="#">Rsh</a>	Allows a user on the client to execute a command on the remote system.

---

## Abort ()

---

### Summary:

.Aborts the command in progress.

### Syntax:

```
Int Abort()
```

### Description:

The Abort method aborts a session to the remote server. This method resets and closes the connection without properly closing down and should not be called under normal circumstances.

The Abort method takes no parameters and returns an integer.

### Return Values:

This method always returns 0.

### See Also:

Rexec () and Rsh () methods

### Example:

```
Rcmd1.Abort();
```

[Return to Remote Commands Method Summary](#)

[Return to Remote Commands Property Summary](#)

## Rexec ()

### Summary:

Logs into the remote system and executes the specified command on that system.

### Syntax:

Int Rexec (String *Host*, String *User*, String *Password*, String *Command*)

Parameter	Meaning
<i>Host</i>	This is the name or IP address of the system on which the command is to be executed
<i>User</i>	This is the name of the user account on the remote system that will be used to allow access to the system.
<i>Password</i>	This is the password for the user account to be logged in. The password is in clear text.
<i>Command</i>	This is the command to be executed on the remote system.

### Description:

The Rexec method is used to execute a single command on a remote machine. First, a connection to the remote machine is established. Then, the command is executed on the remote machine and the result is sent to the application. The connection terminates as soon as it executes the remote command.

If the remote command can be successfully executed, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value

### Warning:

If you are saving the server responses to a file, make sure that you have the correct permissions to create a file in the specified folder.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The method completed successfully
1	ERR_CANNOT_CONNECT	A connection could not be made to the remote host.
2	ERR_HOST_NOT_DEFINED	The host name is not defined
3	ERR_USER_NOT_DEFINED	The user name or password were not specified.
4	ERR_IN_ACTION	An other method is already in progress.
5	ERR_FILE_CREATE	The file to contain the server responses could not be created. Please check the permissions for the specified folder.
6	ERR_WINSOCK_INIT	Winsock could not be initialized.

### See Also:

ReceiveInFile, ReceivedDataPath, ReceivedDataFileName and ReceivedData properties.

### Example

```
//this function performs rexec
function rexec()
{
    //determining the "save to file" option
```

## Distinct Remote Commands COM Object

```
if (tofile.checked)
    Rcmd1.ReceiveInFile=true;
else
    Rcmd1.ReceiveInFile=false;

//calling rexec
var result=Rcmd1.Rexec("HostName", "User", "Password", "Command" );
if (result == 0)
{
    //starting server monitoring thread to display result
    alert ("Command executed successfully");
}
else
    alert ("Error: " + result + "\r\n" + Rcmd1.ErrorMessage);
}
```

[Return to Remote Commands Method Summary](#)

[Return to Remote Commands Property Summary](#)

## Receive ()

### Summary:

Reads the server response and places it into the ReceivedData property or the data file that will contain the server responses if ReceiveInFile property is set to TRUE.

### Syntax:

```
Int Receive ()
```

### Description:

This method can be used to force the Remote Commands Object to read the available data from the socket. This method can be called in a loop till the time CommandCompleted property is set to true.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The method completed successfully
1	ERR_CANNOT_CONNECT	A connection could not be made to the remote host.
2	ERR_HOST_NOT_DEFINED	The host name is not defined
3	ERR_USER_NOT_DEFINED	The user name or password were not specified.
4	ERR_IN_ACTION	
5	ERR_FILE_CREATE	The file to contain the server responses could not be created. Please check the permissions for the specified folder.
6	ERR_WINSOCK_INIT	Winsock could not be initialized.
7	ERR_LICENSE	Check to see if you have a valid license number.
8	ERR_CONNECT_TO_RECV	You must have a valid connection before calling this method.

### See Also:

Rexec () and Rsh () methods.

### Example

```
// this function displays the result of rexec or rsh commands function ShowResult()
{
    if (!Rcmd1.ReceiveInFile)
    {
        //showing the server reply
        srply.value="Server returned:\r\n"+Rcmd1.ReceivedData+"\r\n";

        // check if the server reply is completed
        if (Rcmd1.CommandCompleted) return;
        Rcmd1.Receive ();
        //if the reply received is not complete, continue waiting
        window.setTimeout("ShowResult()",200);
    }
    else
    {
        //if the reply received is not complete, continue waiting
```

## Distinct Remote Commands COM Object

```
if (!Rcmd1.CommandCompleted)
{
    Rcmd1.Receive ();
    window.setTimeout
    ("ShowResult()",200);
    return;
}

//show the filename server reply was saved into
srply.value="Result saved in file: " + Rcmd1.ReceivedDataFileName + "\r\n";
}
}
```

[Return to Remote Commands Method Summary](#)

[Return to Remote Commands Property Summary](#)

## Rsh ()

### Summary:

Executes a command on a remote system.

### Syntax:

Int Rsh (String *Host*, String *User*, String *Command*)

Parameter	Meaning
<i>Host</i>	The name or IP address of the host on which you wish to execute a specified command
<i>User</i>	The user account name to be used to access the remote host.
<i>Command</i>	The command to be executed.

### Description:

The Rsh method connects to the remote host and executes the specified command. The remote user name is required to execute the command. The remote shell connection normally terminates as soon as the remote command is executed.

If the remote command can be successfully executed, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value.

### Return Values:

The following are the values that may be returned:

Value	Error Message	Meaning
0	ERR_SUCCESS	The method completed successfully
1	ERR_CANNOT_CONNECT	A connection could not be made to the remote host.
2	ERR_HOST_NOT_DEFINED	The host name is not defined
3	ERR_USER_NOT_DEFINED	The user name or password were not specified.
4	ERR_IN_ACTION	Another method is in progress.
5	ERR_FILE_CREATE	The file to contain the server responses could not be created. Please check the permissions for the specified folder.
6	ERR_WINSOCK_INIT	Winsock could not be initialized.
7	ERR_LICENSE	Check to see if you have a valid license number.

### See Also:

ReceiveInFile, ReceivedDataPath, ReceivedDataFileName and ReceivedData properties.

### Example:

```
//this function performs rsh
function rsh()
{
    //determining the "save to file" option
    if (tofile.checked)
        Rcmd1.ReceiveInFile=true;
    else
        Rcmd1.ReceiveInFile=false;

    //calling rexec
```

## Distinct Remote Commands COM Object

```
var result=Rcmd1.Rsh("MyHostName", "MyUser", "MyCommand");
if (result == 0)
{
    //starting server monitoring thread to display result
    alert ("Command executed successfully");
}
else
    alert ("Error: " + result + "\r\n" + Rcmd1.ErrorMessage);
}
```

[Return to Remote Commands Method Summary](#)

[Return to Remote Commands Property Summary](#)

---

## 6 – Rlogin Object

### Rlogin Overview

The Rlogin command allows users to log into a remote system, that is running the rlogind, from a Windows workstation.

In order to make a connection using Rlogin 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. Note that different rlogind's may have slightly different security requirements. Check your server documentation for exact information on how to set this up on your particular server.

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 Rlogin successfully:

1. Add the following entries in /etc/hosts file on the Unix system called:

```
unix1.mydomain.com:  
192.0.0.3 mypc mypc.mydomain.com
```

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

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

3. You must also add the host to the /.rshosts for superuser access. Otherwise it needs to be added to the rhosts directory in the user's home directory.

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.

You must also add the host to the /.rshosts

### How to Use the Rlogin Object

To use the Rlogin Object you must embed it in your ASP or HTML page.

### How to embed the control in a HTML page

To embed the Rlogin Object in your HTML page add the following lines:

```
<OBJECT ID="Rlogin1" WIDTH="2" HEIGHT="1"  
  CLASSID="CLSID:E4E5888A-25A0-4A08-BDA6-03DAA9591356">  
  CODEBASE="./dsv_rlogin.cab">  
</OBJECT>
```

## Distinct Rlogin COM Object

Note that you need to create a cab file that contains the dsv\_rlogin.ocx and dsv-rlib.dll files as shown in the sample that comes with this product.

## How to embed the control in an ASP page

To embed the Remote Commands Object in your ASP page add the following line:

```
Set RloginObj = Server.CreateObject ("DistinctServerRlogin.RloginCtrl")
```

## Registry Entries for Rlogin

Various parameters used to control the behavior of the Distinct Rlogin Object are stored under the following registry path:

HKLM\SOFTWARE\Distinct\DLLS\Web\RLIB

The following entries define the settings that can be modified through registry entries:

RloginTimeout REG\_DWORD 1-999

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

RloginSendBuf REG\_DWORD 1024-32000

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

RloginReceiveBuf REG\_DWORD 1024-32000

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

---

## Rlogin Properties

The following is a description of all the properties that can be set in the Rlogin COM Object.

### Property Summary

The following lists all the properties of the Rlogin COM Object and gives a brief summary of their function:

Property	Function
<a href="#">CMDLinePrompt</a>	Used to login the user automatically.
<a href="#">ErrorMessage</a>	Contains the error response from the remote server
<a href="#">PasswordPrompt</a>	Contains the various possible server password prompts.
<a href="#">ReceivedData</a>	Contains the server responses.
<a href="#">ReceivedDataPath</a>	Contains the path to the data file if the server responses are saved to file.
<a href="#">ReceivedDataFileName</a>	Contains the name of the file containing the server responses.
<a href="#">ReceiveInFile</a>	Specifies whether the server responses will be stored in the ReceivedData property or in a file.

---

## CmdlinePrompt

---

### Summary

Contains the possible prompts that the server may issue as the command prompt.

### Description:

This property is used when the Rlogin method is called with a password and command. It contains the various possible server command prompts delimited by commas.

After the connection is made, the Rlogin object tries to match each of the password prompts and the command prompts with the server response. If a match is found with the password prompt, it sends the supplied password to the server and waits for the command prompt. If a match is found with one of the command prompts, it sends the supplied command to the server and assumes that the auto-login is complete.

### Type:

String

### Default Value:

The default value is %, \$

### See Also:

PasswordPrompt property and Rlogin () method.

### Example:

```
function Connect()
{
    var result=-1;

    // set the command prompt and the password prompts
    Rlogin1.PasswordPrompt = "password:";
    Rlogin1.CmdLinePrompt = "#";

    // call the Rlogin method to display the directory listing
    result=Rlogin1.Rlogin("MyHost", "MyUser", "MyPassword", "ls -l" + "\r\n");

    // check the return value
    if (result == 0)
        alert ("Command executed");
    else
        alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## ErrorMessage

---

### Summary

This property contains the error message received from the server in case of a connection or command execution error.

### Description:

The ErrorMessage property contains an error if a connection could not be established. If the command could not be properly executed for some other reason, it will contain the response from the server. This property will contain the exact response from the server.

### Type:

String. Read Only.

### Default Value:

This property has no default value.

### See Also:

Rlogin () method

### Example:

```
function Connect()
{
    var result=-1;

    // set the command prompt and the password prompts
    Rlogin1.PasswordPrompt = "password:";
    Rlogin1.CmdLinePrompt = "#";

    // call the Rlogin method to display the directory listing
    result=Rlogin1.Rlogin("MyHost", "MyUser", "MyPassword", "ls -l" + "\r\n");

    // check the return value
    if (result == 0)
        alert ("Command executed");
    else
        alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## Password Prompt

---

### Summary

This property contains a series of prompts that may be returned by the server as its password prompt.

### Description:

This property is used to automatically log in the user when calling the Rlogin method. It contains the various possible password prompts delimited by commas. After the connection is made, control tries to match each of the password prompts and the command prompts with the server response. If a match is found with the password prompt, it sends the supplied password to the server and waits for the command prompt. If a match is found with one of the command prompts, it sends the supplied command to the server and assumes that the auto-login is complete.

### Type:

String

### Default Value:

The default value is word:

### See Also:

CmdlinePrompt property

### Example:

```
function Connect()
{
    var result=-1;

    // set the command prompt and the password prompts
    Rlogin1.PasswordPrompt = "password:";
    Rlogin1.CmdLinePrompt = "#";

    // call the Rlogin method to display the directory listing
    result=Rlogin1.Rlogin("MyHost", "MyUser", "MyPassword", "ls -l" + "\r\n");

    // check the return value
    if (result == 0)
        alert ("Command executed");
    else
        alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## ReceivedData

---

### Summary

This property contains the server response to the Rexec or Rsh commands.

### Description:

The ReceivedData property is used to access data that has been sent by the server. This property is set in response to the Receive () method.

### Type:

String

### Default Value:

There is no default value for this property.

### See Also:

Rlogin methods

### Example:

```
// this function display the result sent by the server after a command is executed
function srvThread()

{
    Rlogin1.Receive();
    // get the server response read so far
    rstr=new String(Rlogin1.ReceivedData())

    //and display it
    sdlg.value=rstr;

    // If response is incomplete call this function again
    sthread=window.setTimeout("srvThread()",100);
}
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## ReceivedDataPath

---

### Summary

This property is used to specify the path where the data file containing the server responses should be created if the data is to be stored in a file.

### Description:

This property is used to set the path of the file in which the received data is stored if ReceiveInFile property is set to true. User needs to set this property if the default location is not desired.

### Type:

String.

### Note:

When setting this path make sure that there are appropriate write permissions for the file to be created.

### Default Value:

The default value for this property is the current folder for the process that loaded the Rlogin object.

### See Also:

ReceivedDataFileName and ReceiveInFile properties and Rlogin () method.

### Example:

```
function Connect()
{
    var result=-1;

    // set the command prompt and the password prompts
    Rlogin1.PasswordPrompt = "password:";
    Rlogin1.CmdLinePrompt = "#";
    Rlogin1.ReceiveInFile = true;
    Rlogin1.ReceivedDataPath = "c:\temp";
    // call the Rlogin method to display the directory listing
    result=Rlogin1.Rlogin("MyHost", "MyUser", "MyPassword", "ls -l" + "\r\n");

    // check the return value
    if (result == 0)
        alert ("Command executed");
    else
        alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## ReceivedDataFileName

---

### Summary

This property contains the name of the file in which the server responses are stored.

### Description:

If ReceiveInFile property is set to true, this property will be populated by the object with the name of the file where all the received data is saved. The file will be created when the ReceiveInFile property is set to true. The main purpose of this is to receive binary data. Every time a command is executed, the file will be truncated to 0 length so that the new response and old responses do not get mixed up. The same file name will be used to receive all the responses for a single command.

### Type:

String

### Default Value:

There is no default value for this property. This property is read only.

### See Also:

ReceivedDataPath and ReceiveInFile properties and Rlogin () method.

### Example:

```
Alert("Response is saved in file:" + Rlogin1.ReceivedDataFileName)
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## ReceiveInFile

---

### Summary

This property specifies whether the server responses will be stored in a file or in the ReceivedData property.

### Description:

When this property is set to true, all the received data is stored in a file and the file name is returned in the ReceivedDataFileName property. This property is useful in receiving the binary data.

### Type:

Boolean.

### Default Value:

The default value for this property is FALSE

### See Also:

ReceivedDataPath and ReceivedDataFileName properties and Rlogin () method.

### Example:

```
function Connect()
{
    var result=-1;

    // set the command prompt and the password prompts
    Rlogin1.PasswordPrompt = "password:";
    Rlogin1.CmdLinePrompt = "#";
    Rlogin1.ReceiveInFile = true;
    Rlogin1.ReceivedDataPath = "c:\temp";
    // call the Rlogin method to display the directory listing
    result=Rlogin1.Rlogin("MyHost", "MyUser", "MyPassword", "ls -l" + "\r\n");

    // check the return value
    if (result == 0)
        alert ("Command executed");
    else
        alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

[Return to Rlogin Property Summary](#)

[Return to Rlogin Method Summary](#)

---

## Rlogin Methods

This section documents all the exported methods of the Rlogin COM object.

### Method Summary

The following lists all the methods and gives a brief description of their function:

Method	Function
<a href="#">Abort</a>	Aborts the rlogin session.
<a href="#">Disconnect</a>	Disconnects the current session
<a href="#">Receive</a>	Takes the server response and places it in the property or file.
<a href="#">Rlogin</a>	Logs into the remote system and executes a command on the remote system.
<a href="#">Send</a>	Sends an additional command to the remote system.

---

## Abort ()

---

### Summary:

Aborts an Rlogin Session.

### Syntax:

```
Int Abort()
```

### Description:

The Abort method aborts a session to the remote server. This method resets and closes the connection without properly closing down and should not be called under normal circumstances.

The Abort method takes no parameters and returns an integer. This method always returns 0.

### See Also:

Rlogin () method.

### Example:

```
Rlogin1.Abort();
```

[Return to Rlogin Method Summary](#)

[Return to Rlogin Property Summary](#)

---

## Disconnect

---

### Summary:

Disconnects the current Rlogin session.

### Syntax:

```
Int Disconnect ()
```

### Description:

The Disconnect method disconnects a session to the remote server. This method must be called when a connection is no longer needed.

### Return Values:

This method returns one of the following values:

Value	Message	Meaning
0	ERR_SUCCESS	Disconnect was successful
6	ERROR_IN_ACTION	Another command is in progress.

### See Also:

Rlogin () method.

### Example

```
Function DisconnectFromServer()  
{  
    Rlogin1.Disconnect();  
}
```

[Return to Rlogin Method Summary](#)

[Return to Rlogin Property Summary](#)

## Receive ()

### Summary:

This property takes the server response and places it in either ReceivedData property or in a file if the ReceiveInFile property is set to TRUE.

### Syntax:

```
Int Receive ()
```

### Description:

This method can be used to force the control to read the available data from the socket. This method can be called in a loop to get the full reply from the server.

### Return Values:

This method returns one of the following values:

Value	Message	Meaning
0	ERR_SUCCESS	The Data was successfully passed to the property or file.
9	ERROR_CONNECT_TO_RECEIVE	You must first have an Rlogin connection before calling this method.

### See Also:

Rlogin () method.

### Example

```
// this function display the result sent by the server after a command is executed
function srvThread()

{
    Rlogin1.Receive();
    // get the server response read so far
    rstr=new String(Rlogin1.ReceivedData())

    //and display it
    sdlg.value=rstr;

    // If response is incomplete call this function again
    sthread=window.setTimeout("srvThread()",100);
}
```

[Return to Rlogin Method Summary](#)

[Return to Rlogin Property Summary](#)

## Rlogin

### Summary:

This property logs into a remote system using a user name and password and executes a specified command on the remote system.

### Syntax:

Int Rlogin (String *Host*, String *User*, String *Password*, String *Command*)

Parameter	Meaning
<i>Host</i>	This is the name or IP address of the host to log into.
<i>User</i>	This is the user name to be used to log into the system
<i>Password</i>	This is the user's password. This parameter is optional and may be specified by the user after login.
<i>Command</i>	This is the command to be executed on the remote system and can be specified by the user after login.

### Description:

The Rlogin method is used to connect and execute commands on a remote machine. First, a connection to the remote machine is established. Then, the command is executed on the remote machine and the result is sent to the application. The application can execute another command at the server by calling the Send () method. The application should call the Disconnect method to terminate the connection when the connection is no longer needed.

If the remote command can be successfully executed, then the method returns 0; otherwise, it returns an error. The application should ensure that the method was successfully executed by checking the return value.

### Note:

The Receive () method must be called following the Rlogin command to read the data returned by the server and place it in the property or file.

### Return Values:

This method returns one of the following values:

Value	Message	Meaning
0	ERR_SUCCESS	The command was successfully executed.
1	ERR_CANNOT_CONNECT	The object was unable to connect
2	ERR_HOST_NOT_DEFINED	The host name was not specified.
3	ERR_USER_NOT_DEFINED	The user name was not specified.
4	ERR_FILE_CREATE	The file to contain the server responses could not be created. Check permissions.
5	ERR_WINSOCK_INIT	Socket error
6	ERR_IN_ACTION	Another method was in progress.
10	ERR_LICENSE	A licensing error occurred.

### See Also:

ReceivedData, ReceiveInFile, ReceivedDataPath and ReceivedDataFileName properties and Receive () method.

### Example

```
function Connect()
{
```

## Distinct Rlogin COM Object

```
var result=-1;

// set the command prompt and the password prompts
Rlogin1.PasswordPrompt = "password:";
Rlogin1.CmdLinePrompt = "#";

// call the Rlogin method to display the directory listing
result=Rlogin1.Rlogin("MyHost", "MyUser", "MyPassword", "ls -l" + "\r\n");

// check the return value
if (result == 0)
    alert ("Command executed");
else
    alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

[Return to Rlogin Method Summary](#)

[Return to Rlogin Property Summary](#)

## Send ()

### Summary:

Sends a command to the remote system following the server response.

### Syntax:

Int Send (String *Buffer*, Boolean *IsFile*)

Parameter	Meaning
<i>Buffer</i>	String containing the command to be sent or a file containing the data to be sent to the server.
<i>IsFile</i>	When set to TRUE this means that the <i>Buffer</i> contents is a filename. When set to FALSE it is a string to be passed to the server as is.

### Description:

The Send method is used to send a command to the server.

This method is useful if you are creating an application that is interactive. User input in response to the server response can be sent using this method.

The response to these user commands can be read from the ReceivedData property. Please note that the command is sent as it is entered by the user so if the server expects the command to terminate with CRLF, it should be done at the application level. The Object simply sends the contents of the buffer or the file as is.

### Return Values:

This method returns one of the following values:

Value	Message	Meaning
0	ERR_SUCCESS	Send was successful
4	ERR_FILE_CREATE	Error in creating the file that will contain the server response.
6	ERR_IN_ACTION	Another command is in progress.
7	ERR_CONNECT_TO_SEND	You must have an Rlogin connection before you can send a command to the server.
8	ERR_CANNOT_SEND	The Object timed out waiting to send the command.

### See Also:

Rlogin () method.

### Example

```
function Send()
{
    var result=-1;

    // call the send method to display the directory listing
    result=Rlogin1.Send("ls -l" + "\r\n", false);

    // check the return value
    if (result == 0)
        alert ("Command executed");
    else
        alert ("Error: " + result + " " + Rlogin1.ErrorMessage);
}
```

Distinct Rlogin COM Object

}

[Return to Rlogin Method Summary](#)

[Return to Rlogin Property Summary](#)