EZDIALUP™ v1.75

EZDialup v1.75, a Windows® 3.x/95-compatible communications program, provides a reliable but rapid development system when you need...

A. a **complete, automated dialup system** without doing any programming

B. the same capability **integrated into your own programs** with minimal effort

C. to write a program that controls the server PC without using scripts

D. to write a program that dials a BBS or other terminal host (**NEW**)

This document is divided into sections (A-D) that describe in detail the use of each "mode" of EZDialup. Registration information and a version chronology are located at the end of the document.

Mode A

Complete installation instructions for servers and clients are included below. To summarize, the steps required to install a complete system are:

-Install modem(s) in server PC, adjust Windows port settings, and attach phone lines.
-Create a directory for EZDialup on network (or server drive) and install software
-Create a directory for each user and assign passwords.
-Modify supplied sample scripts to meet your needs.
-Install software on remote PCs.

Physical Server Installation

Set up a dedicated Windows PC. If the files to be served are on a network, the PC should be attached to the network and logged in. For each line (up to four), install a Hayes-compatible modem. Obviously, each must occupy different ports (COM1 through COM4). SInce the PC will probably have at least one port (COM1) built-in as a DB9 or DB25 connector, you must either disable the built-in ports or attach external modems to them.

Each modem must also different IRQ's. By default, COM1 is IRQ4, COM2 is IRQ3 and COM3 and COM4 share IRQ's with COM1 and COM2. This sharing of IRQ's ***does not work***, so the internal modems *must* be capable of being changed to use different IRQ's. IRQ5 by default supports the second parallel port and can be used. IRQ10 and IRQ11 are usually open, but remember that the network adaptor that would be needed in this system needs an IRQ, too. IRQ7 supports the main parallel port - if IRQ's get tight, it might be possible to disable LPT1 and use IRQ7.

Windows comm adjustments (multiple modems)

Once the IRQ's and COM's are straightened out, inform Windows about the setup through the Control Panel icon Ports. The Advanced button gets you to the IRQ's. You shouldn't need to change the bases, but here's the info just in case:

COM1	3F8	IRQ4
COM2	2F8	IRQ3
COM3	3E8	IRQ5 or 7 or 10 or 11

COM4 2E8 IRQ5 or 7 or 10 or 11 (but different from COM3)

Test each modem using Windows Terminal. Set it for each COM port (Settings| Communications) and type ATZ and hit ENTER. You should get an OK back or a 0.

Server Software Installation

Create a directory on the network (or server drive) - we'll call it \EZDIALUP. Move all files contained in EZDIALUP.ZIP into this directory. Give the PC authority to the \EZDIALUP and \EZMAIL (if using EZMail) directories and/or any other directories containing data to be distributed.

For each modem, create in \EZDIALUP a text file with the extension .INI (EZDIAL1.INI, EZDIAL2.INI, etc.) modeled after the EZDIALUP.INI supplied in the original EZDIALUP.ZIP file. The .INI files are described in detail below.

Then, create an icon for each modem in the PC's Startup group. The properties of the icons should look like this:

Description:	EZDialup Line 1	<- or Line 2, Line 3, etc.
Command Line:	ezdialup.exe ezdial1.ini	<- or ezdial2, ezdial3, etc.
Working Directory:	k:\ezdialup	<- This must be the full path

Check the "Run Minimized" box.

Note: DO NOT include the complete path of the .INI after "ezdialup.exe"; let the Working Directory indicate the path where *all* the required files can be found.

Attach phone lines to the modems. The lines should be configured in a "circle hunt" (i.e. if a line is busy, try the next one). The server is ready to answer calls.

Server Scripts

Communication scripts control the data transfers that occur during a call. When client calls, it passes a small file that specifies:

-A directory (unique for each remote user)
-A password (checked against the file COMMPASS in this directory)
-A script file name (a text with a .LST extension)

The only real work involved in setting up a repetitive data transfer is in creating this script using the guidelines that follow. It's usually easiest to copy examples and adjust them to each specific instance.

A script file is simply a text file with a .LST extension, and since it's a text file it can be created or modified with NOTEPAD or any other plain text editor. The name doesn't matter, but must match the file name (and be in the directory) specified in the .EZD file sent by the client side (see **Client Software Setup** below).

Each line in a script contains a command line and its parameters. Upper case, lower case or mixtures are acceptable. Lines that start with ; are ignored to let you add comment lines.

Script commands:

DOWNLOAD k:\somedir\somefile.dat c:\otherdir\otherfil.dat
Copy a file from the server to the client. The first parameter is the server-side file path,

the second is the client-side (target) file path. If the client-side file is already there, it is overwritten.

UPLOAD c:\somedir\somefile.dat k:\otherdir\otherfil.dat
Copy a file from the client to the server. The first parameter is the client-side file path, the second is the server-side (target) file path. If the server-side file is already there, it is overwritten.

MOVEDOWN k:\somedir\somefile.dat c:\otherdir\otherfil.dat
Move a file from the server to the client. If the file transfer is complete the file on the server side is deleted. If the file on the client side already exists the transfer is cancelled and the server-side file preserved (not deleted).

MOVEUP c:\somedir\somefile.dat k:\otherdir\otherfil.dat
Move a file from the client to the server. If the file transfer is complete the file on the client side is deleted. If the file on the server side already exists the transfer is cancelled and the client-side file preserved (not deleted).

UNZIP c:\somedir\somefile.zip c:\otherdir client
Un-zip the files stored in the .ZIP file listed in first parameter into the directory in the second parameter. Files are overwritten if already there. This occurs on the client side if the third parameter is "client" or the server side if it's "server". A typo would default to "client".

ZIP k:\somedir\somefile.zip k:\otherdir*.* server
Add to the .ZIP file listed in the first parameter the files that match the path/wildcard in the second parameter. The file is *not* overwritten if already there - use the DELETE command (below) to start a new one. If the file is already in the .ZIP file, and the date/time stamp is the same, the file is not added again; but if the file is newer it *will* replace the older version.

This occurs on the server side if the third parameter is "server" or the client side if it's "client". A typo would default to "server".

DELETE c:\somedir*.dat client
Delete the files that match the path/wildcard. This occurs on the client side if the second parameter is "client" or the server side if it's "server". A typo would default to "client". DELETE can erase a single file or, using the asterisk (see example), a group of files.

EXECUTE "c:\somedir\aprogram.exe someparam anotherparam" nowait server
Run a program. Quotes are required only if parameters are used. The "nowait" means that the rest of the script can continue to run while this program runs - change it to anything else (like "wait") to make the script wait until the program finishes before continuing. The last parameter, "server", means that the server side will run the program - anything else (like "client") causes the program to run on the client. Both DOS and Windows programs can be executed.

UPDATE c:\somedir k:\otherdir c:\somedir\somefile.dat
Note that this one has three parameters and is a bit more complicated. It looks in the server-side directory (the second parameter) for files newer than the client-side file (the third parameter). Sub-directories are included in the search.

All matching files are zipped together, the zip file downloaded to the client, and the zip file is unzipped into the client-side directory in the first parameter with the directory structure kept intact.

The theory behind this function? Copy an entire directory structure from the server side to a client. Files on the server side can then change and new files can be created - the time-and-date (T.A.D.) stamps for these files will be updated.

When the client calls in for an update, EZDialup can take the T.A.D. stamp of a selected file (let's call it the "time" file, essentially the "new-ness" of the client-side data), gather all server files that are newer, then transfer them to the appropriate directories on the client PC.

One of these newer files will be the server version of the "time" file, so the client-side "time" file will have a new time-and-date stamp. If the user were to immediately update again, the server side would find no files newer and issue an "Up to date" message to the client.

To make this work, the time-and-date stamp of the control file on the server must be given a new T.A.D. stamp every time the a file in the directory structure is changed or created. The only downside to overlooking this rule is that the client-side software would not realize how "new" and up-to-date it is the next time it calls and might end up retrieving some files unnecessarily.

Note that we've specified that changing or creating files will result is new time-and-date stamps. If you copy an existing file into the structure, the T.A.D. will be the same as the original, which might not be newer than the "time" file.

MAIL k:\ezmail\abc.box c:\ezmail\mail.zip
Perform a mail update. Works only with EZ Software's EZMail. The first parameter (k:\ ezmail\abc.box) is the location and name of the user's mailbox on the server side. The second parameter is the location of EZMail on the user's remote PC.

Server History Files

Server-mode copies of EZDialup maintain a history of calls in text files that have .HST extensions. Since each server-mode copy has it's own .INI file (EZDIAL1.INI, EZDIAL2.INI, etc.), the name automatically assigned to each history file is the .INI name with a .HST extension (EZDIAL1.HST, EZDIAL2.HST, etc.). You can occasionally delete these files if they grow too large.

Client Software Setup

Create an \EZDIALUP directory on the client PC. Copy to this directory all the files in EZDIALUP.ZIP, or included only the minimal files required:

EZDIALUP.EXE
UZDLL20.DLL
ZDLL20B.DLL
ZDLL20A.DLL
BWCC.DLL
CLIENT.INI (Name's not important, of course, but you need an .ini file)
SCRIPT1.EZD (An .ezd file is needed for each script the client will request)

For each type of dialup session you want make available for a user, create an icon with properties as follows:

Description:	Get Data	<- whatever the icon will do
Command Line:	ezdialup.exe script1.ezd	<- or another .EZD file
Working Directory:	\ezdialup	<- This must be the full path

In this example, you must then create a file named SCRIPT1.EZD in the \EZDIALUP directory. This is a text file, with one line, which looks like this:

LINK k:\somedir password script1.lst

The first parameter is the remote user's unique directory (on the server side), which contains the password file (a text file called COMMPASS) and the script files. The second password must match the file COMMPASS. For any given nremote user, the first two parameters for all .LST files are the same. The third parameter is the name of the script to execute.

If you do not specify an .EZD file, EZDialup defaults to the name EZDIALUP.EZD

All other parameters are controlled via the file EZDIALUP.INI. The .INI files are described in detail below.

Each icon can have it's own .INI file, if desired. To use an .INI called CLIENT2.INI, for example, the Command Line listed above would be changed to:

ezdialup.exe client2.ini script1.ezd

Make sure the .INI file name is listed *before* the .EZD file name,

Client Software Usage

The remote caller simply runs an EZDialup icon to place a call. If the user notices that the phone number dialing sequence needs to change (most likely due to placing the call from a different place), they can press the "Change Phone Number" button. If they type in a new number and press OK, EZDialup will start a new call with the new number.

EZDialup will hangup and shut down after the data update. If any type of problem occurred, a message box informs the user after the hangup.

Security:
The preceding explanation describes the use of simple text files to hold scripts, passwords, script requests, etc. The supplied program SCRAMBLE.EXE can convert a simple text file into an encrypted one. Client-side .EZD files (which contain passwords) should be scrambled before they are supplied to the remote user. The files called COMMPASS in each user's directory (the actual passwords) should also be scrambled. Finally, scramble the .LST script files.

The supplied program UNSCRAMB.EXE will convert a scrambled text file into a readable one.

Registered users may request an alternative encryption program that is not distributed to the public domain.

.INI file controls

SERVER = TRUE
This line makes EZDialup act as a server. Change to "Server = False" for client operation.

PHONE = 9,555-555-5555
Client-side only. This is the phone number the modem will dial. Commas denote a two-second pause that is often needed when a prefix (like 8 in this example) is used to get an

outside line. The .INI file can contain several lines that start with "PHONE" as long as all but one actually start with a ; (i.e. a comment line). That way a roaming user (home, hotel, HQ, etc.) need only "un-comment" the phone number needed. Put this line at the top - it will be the only line the user will need to change after initial setup.

COMMSTRING = 19200,N,8,1
This is the actual string passed to Windows to initialize the communications port. Have tested it yet, but Win95 may support 38400,N,8,1 and 28,800 modems. Please let us know if it does!

Sometimes very poor telephone connections may necessitate lowering the 19200 number to 9600. Again, you can actually leave two COMMSTRING lines in the file and comment out the 9600 line. The comment lines also let you build an explanation right into the file.

COMMPORT = COM2
Designates the communications port to use. To have EZDialup attempt to auto-locate a modem, change this to COMMPORT = COMX.

INIT1 = ATZ
The first initialization string sent to the modem. Just a reset. In some modems, there may be a distinction between ATZ (Reset) and AT&F (Return to factory defaults). Try both.

INIT2 = ATE0V0S0=0X0
(or INIT2 = ATE0V1S0=0X4 < for client)
This is the minimal modem setup string. For the server side, it may be beneficial to find the codes for your modem that enable error correction and data compression and add them to the end. The codes that are shown usually have the following meanings:

AT simply starts the command.

E0 asks the modem *not* to repeat every command back to the computer.

V0 (Server side) asks modem to report situations with a single-digit number (like 3 for "Connected!"). '

V1 (Client side) asks modem to report situations with phrases (i.e. "CONNECT 14400/V.32 bis").

X0 (Server side) asks modem to report connection with a 3 regardless of the sophistication of the connection (error correction, etc,),

X4 (Client side) asks modem to report connection with all available detail.

WINDOWTITLE = A Window Title
The title bar of the EZDialup window can be changed by adding this line.

ERRORTOLERANCE = 20
This (20) is the default. Specifies the number of bad blocks encountered before cancelling call.

BLOCKSIZE = 4000
Specifies the size of data blocks sent during downloads. Numbers higher that 4096 will be rounded down to that number.

UPLOADBLOCKSIZE = 4000
Specifies the size of data blocks sent during uploads. Numbers higher that 4096 will be rounded down to that number.

TIMERINTERVAL = 200
In this example, every 200 milliseconds (2/10's of a second), EZDialup checks for incoming data during file transfers. Changing downward would have little positive effect. Changing upward may reduce overall system overhead on the server side. Windows is reported to sometimes stop sending communications-oriented messages to an application and therefore a timer is needed to ensure that all data that comes in is received.

SERVERSTATS = FALSE
By default, file transfer stats are not displayed on the server side, but you can change this to ServerStats = true if needed.

CLIENTSTATS = TRUE
By default, file transfer stats are always displayed on the client side, but you can change this to ClientStats = false if desired.

Mode B

First of all, please read **MODE A**. The only difference between the two modes is that in **MODE B** the EZDialup software runs invisibly and sends status messages to your application.

Client Side

To use **MODE B**, your program need use only the first four procedures in the APPDIAL.DLL. In summary, your application needs to:

-Use StartSession() to begin the call, passing your app's window handle
-Process status messages sent to your app by EZDialup during the session
(Useful information for Visual Basic developers can be found below....)
-Provide user a means of cancelling the session, which your app processes by simply calling another routine in the .DLL

The status messages sent by EZDialup can be either re-displayed by your app without modification, or trapped and altered. Some of the messages *must* be trapped, since they provide notification that the session has ended, either normally or due to a problem.

The routines your app willl use in this mode are:

```
procedure StartSession(CmdLine:Pchar;ParentWindow:hwnd);
procedure AbortSession;
procedure  ChangePhoneNumber;
procedure StopEZDialup(ParWindow:hwnd);  (Most likely not required for client side)
```

The CmdLine string in the first routine is exactly what you might use as a command line for standard **Mode A** EZDialup:

```
"C:\EZDIALUP\CLIENT1.INI C:\EZDIALUP\SCRIPT1.EZD"
```

You can also ensure that EZDIALUP.EXE is located by including the complete path of this file at the beginning of the command line:

"C:\EZDIALUP\EZDIALUP.EXE C:\EZDIALUP\ CLIENT1.INI C:\EZDIALUP\..."

Whatever EZDIALUP finds in the .INI file (COM Port, Modem inits, Dialing sequence, etc.) is used during this session, and of course the .EZD file controls which script is requested once a connection is established.

EZDialup runs invisibly, but sends messages to the window handle that was passed as the second parameter of the "Start" routine. This means your app can let the user perform other functions while the telecomm session takes place.

Your app must be set up to process messages sent to WM_USER + 145 (1024 + 145), and additionally but optionally can process messages sent to 146 thru 150. The messages at 145 can be simply re-displayed without modification, but your app will needs to watch for certain phrases to detect the normal completion or abnormal termination of the session. The 32-bit lparam portion of the message is a pointer to a null-terminated string.

The 16-bit wparam portion of the message is an index into a table of these status strings, which allows checking for event occurances without scanning for entire strings. The complete, alphabetical listing can be found on the **Message Listings** section near the bottom of this document.

The phrase "EZDialup Shutdown" is issued your app to let it know everything is finished. If none of the following phrases come in first, the session was completely successful:

28 Error in Opening File:
67 Too many errors. Line quality may be bad
15 Connection could not be established.
5 Cancel Requested (This one means the user requested a shutdown)
82 Your information is already current. (Issued by UPDATEcommand)
65 This session did not end normally.
40 Line Busy - Shutting Down...
44 No Dialtone - Shutting Down...
13 Connect Failed - Shutting Down...
42 Modem Not Initialized.
18 Could not find modem

Your program should also provide it users with a button or some other means of aborting the session - your program then deals with this event by calling the AbortSession() routine listed above.

The third routine, ChangePhoneNumber(), exists to give your app access to the "Change phone number" button offered by the **Mode A** EZDialup program.

The last routine, StopEZDialup(), is provided to give your app the ability to make EZDialup release the communications port, stop running and remove itself from memory. EZDialup will do this automatically after both successful and problem sessions, however, so your app will probably never need to call this function.

The messages that come in at WM_USER + 146 are not critical but can be displayed by your app to give the user additional information about the session. If your app receives an empty (zero length) string here, it should clear the text display it has assigned to this message.

The messages that arrive at WM_USER + 147, 148, 149 and 150 are, respectively, total bytes, elapsed time, throughput (BPS) and percentage complete of the current download. When a download ends, empty strings (zero length) are sent, so if your app displays these messages it should watch for empty strings and clear the text display area assigned to each respective message.

Two responsibilities your app should assume (and can implement using the same flag):
1) do not start an EZDialup session when one is already in progress
2) do not allow your app to shut down when EZDialup is still in progress.

Server Side

Only two of the routines in APPDIAL.DLL are needed to make the server software run invisibly and send message to your program:

1) StartSession(CmdLine:pchar;ParWindow:hwind);

This works exactly like the **client side** - the CmdLine parameter must contain the entire command line you use the start the server, like:

```
"c:\ezdialup\ezdialup.exe c:\ezdialup\server1.ini"
```

The second parameter, ParWindow, must contain the window handle of the program that starts the server. Status messages are then sent to the "parent" program using this handle in the exact same manner as the **client side.**

The server side, in previous versions, has not displayed file transfer statistics on its window, attempting to be as efficient as possible with shared system time. While this is still the default, you can add the line...

```
ServerStats = true
```

...to the .INI file controlling the server to enable the sending of these statistics to your program at messages wm_user + 147 thru 150 - again, see the documentation for details regarding the use of these messages by your program.

The other routine from APPDIAL.DLL your program would use is:

2) StopEZDialup(ParWindow);

Your program can use this routine to remove the server software from memory. The ParWindow parameter must contain the same window handle used when StartSession() was called to start the software. If your program has started multiple servers, it must remove each one separately.

The server-side has been designed to allow a MDI (multiple-document interface) program to easily control multple servers on one PC. Each child window, when created, calls...

```
StartSession('ezdialup.exe serverX.ini',ThisWindow)
```

...passing its own window as ThisWindow, and using a different .INI file (primarily specifying different com ports) for each one.

Then, when the entire program is shutdown and the child windows clean up after thenselves, they must call...

StopEZDialup(ThisWindow)

...again using its own window for ThisWindow, to have EZDialup.exe remove itsef from memory.

Visual Basic information

A few points of interest:

- In order to receive the messages described above, you will need to either write a message blaster or obtain one. MSGBLAST.ZIP is available on Compuserve and may be registered for around $25, though this is someone else's product and the price is subject to change.

- EZDialup sends status messages via 4-byte pointers to null-terminated strings. The following document excerpt from the Microsoft Knowledge Base explains the use of the lstrcpy Windows API function to convert this pointer to a Visual Basic string....

```
Document Number: Q78304          Publ Date: 21-JUN-1995

Because Microsoft Visual Basic does not support a pointer data
type,  you cannot directly receive a pointer (such as a LPSTR) as
the return  value from a Windows API or DLL function.

You can work around this by receiving the return value as a long
integer  data type. Then use the lstrcpy Windows API function to
copy the returned  string into a Visual Basic string.

MORE INFORMATION
================

This information is included with the Help file provided with
Microsoft  Professional Toolkit for Visual Basic version 1.0,
Microsoft Visual Basic  version 2.0, and Microsoft Visual Basic
version 3.0.

An LPSTR Windows API data type is actually a far pointer to a
null-terminated string of characters. Because LPSTR is a far
pointer, it can be received as a four byte data type, such as a
Visual Basic long integer. Using the Visual Basic ByVal keyword,
you can pass the address stored in a Visual Basic long integer
back to the Windows API lstrcpy routine to copy the characters at
that address into a Visual Basic string variable.

Because lstrcpy expects the target string to be long enough to
hold the source string, you should pad any Visual Basic string
passed to lstrcpy to have a size large enough to hold the source
string before passing it to lstrcpy. Failure to allocate enough
space in the Visual Basic string may result in an Unrecoverable
Application Error (UAE) or general protection (GP) fault when you
call lstrcpy.

The following is an example program that demonstrates how to use
lstrcpy to retrieve an LPSTR pointer returned from the Windows API
GetDOSEnvironment routine.
```

```
NOTE: The capability of the Windows API GetDOSEnvironment routine
is already available through the Environ function built into
Visual Basic.  Therefore, the program is useful only to
demonstrate how to use lstrcpy.

'*** General declarations ***
Declare Function GetDosEnvironment Lib "Kernel" () As Long

' Enter the following Declare statement as one, single line:
Declare Function lstrcpy Lib "Kernel" (ByVal lpString1 As Any,
   ByVal lpString2 As Any) As Long

'*** Form Click event code ***
Sub Form_Click()
   Dim lpStrAddress As Long,  DOSEnv$

   ' Allocate space to copy LPSTR into
   DOSEnv$ = Space$(4096)

   ' Get address of returned LPSTR into a long integer
   lpStrAddress = GetDOSEnvironment()

   ' Copy LPSTR into a Visual Basic string
   lpStrAddress = lstrcpy(DOSEnv$, lpStrAddress)

   ' Parse first entry in environment string and print
   DOSEnv$ = Trim$(DOSEnv$)
   DOSEnv$ = Left$(DOSEnv$, Len(DOSEnv$) - 1)
   Form1.Print DOSEnv$
End Sub
```

Mode C

This is a more powerful and convenient way for programmers to use EZDialup's client/server capability. Using this mode, your program does not have to deal with the script or .INI files that normally control everthing in **Modes A & B**).

Everything is accomplished via calls to APPDIAL.DLL; your program can retain the connection to the server as long as it (or the user) needs to; it can react to the results of requests to execute commands (for example, what do if a download request fails); it can request that the client or server machine execute another program and optionally be notified when this program terminates. Your application can request all the commands available in **Modes A & B** (up- and download, zip, unzip and delete files, run programs, and more).

A demo application (built with, and demo-ing the use of, the toolkit) is included, complete with Pascal source code. Here's a **quick shortcut to testing:**

Install all EZDialup files into a directory on the PC that will act as server. Edit the file EZDIALUP.INI and ensure that the line COMMPORT = COM2 is adjusted to match the server's modem, if necessary. You can also change it to COMMPORT = COMX to have EZDialup locate a modem automatically, in which case EZDialup will also

change EZDIALUP.INI to remember the port it located.

Create, with a text editor, a file named COMMPASS (no extension), and type into the top line a password. Save this file, then do a SCRAMBLE COMMPASS. Run EZDIALUP.EXE, and the server should immediately set itself to answer calls.

On the client machine, install all the files, then run TESTBED.EXE. Pull down Connection, then Specify Parameters. Change the phone number to that of the server, the comm port to match the client, the user directory to match the server's directory (or wherever COMMPASS is to be found), and change the password to match COMMPASS on the server.

The client software is now ready to link to the server and execute commands. The Pascal source of TESTBED.EXE is included in EZDIALUP.ZIP.

Check out the **Server Installation** section at top of this document for additional info on setting up the dialup server.

To use Mode C, your program can use the functions and procedures indexed 5-32 in APPDIAL.DLL, plus the AbortSession() procedure (index 2). Call SetParentWindow() as soon as your program starts, and call the procedures indexed 6 through 15 before using any of the others. Detailed explanations of each are listed in the file DIALUNIT.PAS and in EZDIALUP.HLP, and TESTBED.PAS contains the source code for the demo program TESTBED.EXE.

```
procedure AbortSession;                                          index 2

procedure SetParentWindow(ParWindow:hwnd);                       index 5
procedure SetDialingSequence(DialStr:pchar);                     index 6
procedure SetDialupCommPort(PortStr:pchar);                      index 7
procedure SetDialupCommConfig(CfgStr:pchar);                     index 8
procedure SetModemInit1(InitStr:pchar);                          index 9
procedure SetModemInit2(InitStr:pchar);                          index 10
Procedure SetDownloadBlockSize(Size:integer);                    index 11
Procedure SetUploadBlockSize(Size:integer);                      index 12
procedure SetLinkUserPath(path:pchar);                           index 13
procedure SetLinkUserPassword(password:pchar);                   index 14
procedure SetExecutablePath(path:pchar);                         index 15
procedure EstablishDialupLink;                                   index 16
function StartDownload(ServerSource,ClientTarget:pchar):word;  index 17
function StartUpload(ClientSource,ServerTarget:pchar):word;      index 18
function StartMoveDown(ServerSource,ClientTarget:pchar):word;    index 19
function StartMoveUp(ClientSource,ServerTarget:pchar):word;      index 20
function UnzipServerFile(ZipFilePath,TargetServerDir:pchar):word;  index 21
function UnzipClientFile(ZipFilePath,TargetClientDir:pchar):word;  index 22
function ZipServerFile(TargetZipFile,SourcePathandFileMask:pchar):word;  index 23
function ZipClientFile(TargetZipFile,SourcePathandFileMask:pchar):word;  index 24
function DeleteFilesOnServer(SourcePathAndFileMask:pchar):word;  index 25
function DeleteFilesOnClient(SourcePathAndFileMask:pchar):word;  index 26
function RunProgramOnServer(ProgramPath:pchar):word;             index 27
function RunProgramOnClient(ProgramPath:pchar):word;             index 28
function LaunchProgramOnServer(ProgramPath:pchar):word;          index 29
function LaunchProgramOnClient(ProgramPath:pchar):word;          index 30
function UpdateClientDirectory(ClientDirectory,
                    ServerDirectory,
                    ClientDateFilePath:pchar):word;              index 31
```

function EZMailUpdate(ServerMailboxPath,
ClientMailDirectory:pchar):word; index 32

Your app must be set up to process messages sent to WM_USER + 145 (1024 + 145), thru 152.

The messages at 145 can be re-displayed without modification, but your app will need to monitor these messages to determine the status of the dialup session.

The 32-bit lparam portion of the message is a pointer to a null-terminated string. The 16-bit wparam portion of the message is an index into a table of these status strings, which allows checking for event occurances without scanning for entire strings. The complete, alphabetical listing can be found on the **Message Listings** section near the bottom of this document.

Visual Basic(tm) developers can find additional information regarding the conversion of null-terminated strings to Basic strings in the **Mode B** section.

The messages that come in at WM_USER + 146 are not critical but can be displayed by your app to give the user additional information about the session. If your app receives an empty (zero length) string here, it should clear the text display it has assigned to this message.

The messages that arrive at WM_USER + 147, 148, 149 and 150 are, respectively, total bytes, elapsed time, throughput (BPS) and percentage complete of the current download. When a download ends, empty strings (zero length) are sent, so if your app displays these messages it should watch for empty strings and clear the text display area assigned to each respective message.

The messages that arrive at WM_USER + 151 indicate the completion of a requested command. When your app requests a command it receives as a return value a number that the app should consider to be a serial number for the command. When the command is completed this message is sent to your app, along with the command serial number and a pointer to a null-terminated string the indicates the completion status of the command.

The messages that arrive at WM_USER + 152 contain the names of files that the zip functions have zipped or unzipped, which your app can display to give your user something to look at during long zips/unzips.

The minimum files you must include with your app are:

EZDIALUP.EXE << May be renamed - see below
UZDLL20.DLL
ZDLL20B.DLL
ZDLL20A.DLL
APPDIAL.DLL
BWCC.DLL

EZDIALUP.EXE may be renamed, as long as the extension remains .EXE. Do not use the name APPDIAL.EXE.

You can save time during the debugging stage of your client application by letting EZDialup retain the connection to the server during recompiles of your app. If the app calls SetParentWindow() as soon as it begins it will re-link to the EZDialup control

software, which removes itself from memory only after a disconnect request.

Responsibilities your app must assume:

1) If your app has started EZDialup, then requested a hangup, do not attempt to restart EZDialup again until the app receives the string "EZDialup Shutdown" via the messages at WM_USER + 145. If it does, the request for a re-connect is ignored.

2) EZDialup client contains no timeouts - it will not disconnect unless told to do so. You may want to have your app request a hangup upon exiting, which is OK to do even if no connection ever occurred.

3) Your app must use routines indexed 5 thru 15 (i.e. give EZDialup its configuration) before running the EstablishLink command or requesting a command that will establish a link automatically.

Mode D

You can use the **Mode D** extensions to simplify the coding of any program that needs to call a BBS, Unix host, CIS - any dialup host. A complete list of the pertinent API routines can be found at the send of this section.

Your program specfies the dialing sequence, comm port, port configuration and modem inits, then requests a link. You can pre-program the login, password, and other responses to expected events, receiving notification at each step. You can start built-in file transfers that use standard protocols. You have access to all input characters, if you need it, and can also send characters separately or in strings.

It's this easy...

Run **SetParentWindow(hwnd)** when your program starts, passing your program's window handle. Be sure your program runs this routine ***only once.***

Connecting...

To prepare to call, run these routines (change the parameters as needed, of course):

SetDialingSequence("9,555-1212"**);**
SetDialupCommPort("com2"**);**
SetDialupCommConfig("19200,n,8,1'**";**
SetModemInit1("ATZ"**);**
SetModemInit2("ATE0S0=0V1X4"**); << V1X4 lets all Hayes-type modems work**

Rename, if you want, EZDIALUP.EXE to MYAPP.EXE, or whatever, anything but APPDIAL.EXE (since that's the dll name) - some folks prefer to lose the "EZ" and love this option...

SetExecutablePath("c:\myapp\myapp.exe");

Your program is now ready to start the dialup session with...

EstablishLinkAsTerminal

Status Messages...

Your program **must** be set up to receive messages sent to WM_USER + 145 (1024 + 145). The 32-bit lparam portion of the message is a pointer to a null-terminated string. These strings contain status messages that relay to your program the minimal information it needs to know what's occuring during the dialup session. These messages are best demonstrated in the included program, TESTBED.EXE.

Visual Basic(tm) developers can find additional information regarding the conversion of null-terminated strings to Basic strings in the **Mode B** section.

Auto-responses...

You can use **SetupNotification**(...) to pre-program auto-responses when expected strings occur in the input stream...

```
FirstNotifyIndex = SetupNotification("Host Name:","Your Host",0,0);
SecondNotifyIndex = SetupNotification("UIC:","Your UIC",0,0);
ThirdNotifyIndex  = -SetupNotification("Connected","",0,0);
```

or maybe...

```
LoggingNow = SetupNotification("Login: ","Your ID",0,0);
PasswordNow = SetupNotification("Password: ","password",0,0);
NewMail  = SetupNotification("new mail ","",0,0);
```

In the examples above, FirstNotifyIndex, SecondNotifyIndex and ThirdNotifyIndex (or LoggingNow, PasswordNow and NewMail) are variables in your program. They would end up with the values 1, 2 and 3 - an automatically-incremented index. This is true as long as the third parameter is 0 - you can call **SetupNotification** later (using in the third parameter the index number you stored, instead of) to override a notification previously set up, typically to disable it after it's occured. For example, **SetupNotification**("","",FirstNotifyIndex,0);

EZDialup watches for the strings in the first parameter ("Host Name:", etc.) as it passes data to your program, and when it encounters one of the strings it sends a notification message to your program (including the notification index number - 1,2,3 etc.). It also (optionally) automatically sends to the host the string in the second parameter. To disable the auto-response but still receive a notification message, use a string with no length (see the ThirdNotifyIndex examples above) in the second parameter .

Auto-response notification messages sent to your program...

Messages are sent to WM_USER + 161 (1024 + 161) unless the last parameter in **SetupNotification** is non-zero (call it 170, for example) - in this case notification for that particular search string will be sent to WM_USER + 170. This gives you the option of establishing different "triggers" for different notifications, instead of having them all go to 161 and checking the index number (contained in **msg.wparam**) - either way works equally well.

In all notification messages, **msg.wparam** contains the index number of the notification, which lets your program keep a clear picture of what's transpired during the session.

Auto-responses ***will only work if*** your program responds to all messages it receives at WM_USER + 160 - read the next section...

Receiving characters...

Auto-responses ***will only work if*** your program responds to all messages it receives at WM_USER + 160 (some data is waiting) by calling **GetSerialIByte()**. The value found in

the 16-bit wparam portion of the message (**msg.wparam)** tells your program how many bytes (characters) of data are waiting, and therefore how many times you need to loop through calling **GetSeriallByte()...**

```
if  msg.wparam <> 0 do
  for i = 1 to msg.wparam do GetSeriallByte(b); << msg.wparam=bytes waiting
```

Another method you can use to deal with this message (WM_USER + 160)...

```
while SeriallOWaiting do GetSeriallByte(b)
```

What you do with the bytes that come in depends on what state your program's in. If you use auto-response, a lot of the time your program will call **GetSeriallByte()** and just ignore it the data it receives.

Disabling/Enabling Notifications...

You can use **DisableAllNotifications** and **ReEnableAllNotifications** to, respectively, temporarily suspend notifications (even if a search string occurs) and then allow them again. Simply there as a convenience in case the need should arise in your program.

Sending characters...

You can use **SendSerialByte()** and **SendSerialString()** to send characters to the host manually when necessary. **SendSerialString()** will *not* add a carriage-return character automically - concatenate one onto the end of the string if you need to.

File Transfers...

You can automatically upload and download files using the XMODEM, XMODEM 1K and YMODEM file transfer protocols. Use YMODEM if the host supports it, or use XMODEM 1K, or the slowest, plain XMODEM.

Your program firsts sends the appropriate command to the host to begin its side of the transfer, either manually or by setting an autoresponse...

```
        SendSerialString("sb somefile.dat")
        SendSerialByte(chr(13))
  or
        s = "sb somefile.dat" + chr(13)
        SetupNotification("Prompt >",s,0,0)
```

...and when it receives a response that indicates the host is starting the transfer, for example...

```
        SetupNotification("File Transfer Started","",0,170)
```

...and message WM_USER+170 is triggered then execute...

```
        StartTerminalDownload("c:\myapp\somefile.dat",3)  (3 means YMODEM)
      or
        StartTerminalUpload("c:\myapp\somefile.dat",3)
```

Here are the transfer protocol codes...

1 XMODEM
2 XMODEM 1K
3 YMODEM

A note about the FilePath parameter (the first one) of **StartTerminalDownload:** when using one of the Xmodem download protocols, which transfer at most one file at a time, simply supply a complete path and file name. When using Ymodem, you have some options. You can set up the host to transfer multiple files via Ymodem, then execute, for example...

StartTerminalDownload("c:\myapp\data\",**3**)

...which would cause all files downloaded to end up in c:\myapp\data. Make sure the last character in the path is a back-slash ("\") to let EZDialup know this is what you want done. You can also do this if downloading only one file.

You can interrupt a file transfer in progress by using...

InterruptFileTransfer

During file transfers, input characters will *not* be sent to your program, but are instead processed directly by EZDialup. When the transfer ends, your program is sent the message WM_USER+160 (meaning some data is waiting), but the "number of bytes" value (msg.wparam) will be zero.

During the progress of *all* file transfers, your program receives simple progress messages at WM_USER+145 ('Received Block 1", "Transfer Complete",etc.).

During *most* transfers, your program also receives messages at WM_USER+147, 148, 149 and 150. The lparam portion of the messages points to null-terminated strings that show the total bytes (147), elapsed time (148), bytes-per-second (149) and percentage complete (150). The exception: XMODEM downloads - when this protocol must be used, client does not know final file length until the transfer ends.

When a file transfer ends, empty strings (zero length) are sent, so if your app displays these messages it should watch for empty strings and clear the text display area assigned to each respective message.

Connection directly to modem...

Instead of setting a dialing sequence and using **EstablishTerminalLink**, you can immediately connect to the modem by running, for example...

SetDialupCommPort("com2"**);**
SetDialupCommConfig("19200,n,8,1'**";**

then

EstablishCommPortLink

Notification of input characters (as described above in **Receiving Character)** begins immediately, and both **SendSerialString**() and **SendSerialByte**() are also active immediately.

Ending the call...

You can use...

AbortSession

...to hangup and remove EZDialup froim memory.

Minimun Files Listing...

EZDIALUP.EXE << May be renamed - see below
UZDLL20.DLL
ZDLL20B.DLL
ZDLL20A.DLL
APPDIAL.DLL
BWCC.DLL

EZDIALUP.EXE may be renamed, as long as the extension remains .EXE. Do not use the name APPDIAL.EXE.

Responsibilities your app must assume...

1) If your app has started EZDialup, then requested a hangup, do not attempt to restart EZDialup again until the app receives the string "EZDialup Shutdown" via the messages at WM_USER + 145. If it does, the request for a re-connect is ignored.

2) EZDialup client contains no timeouts - it will not disconnect unless told to do so. You may want to have your app request a hangup upon exiting, which is OK to do even if no connection ever occurred.

API routines (APPDIAL.DLL - only those used in Mode D...)

procedure AbortSession; index 2

procedure SetParentWindow(ParWindow:hwnd); index 5
procedure SetDialingSequence(DialStr:pchar); index 6
procedure SetDialupCommPort(PortStr:pchar); index 7
procedure SetDialupCommConfig(CfgStr:pchar); index 8
procedure SetModemInit1(InitStr:pchar); index 9
procedure SetModemInit2(InitStr:pchar); index 10

procedure SetExecutablePath(path:pchar); index 15

procedure EstablishLinkAsTerminal; index 33
Function SerialIOWaiting:boolean; index 34
Function GetSerialByte:integer; index 35
function SendSerialByte(SendByte:byte):boolean; index 36
function SendSerialString(Sendstr:pchar):boolean; index 37
function SetupNotification(SearchStr,ResponseStr:pchar;Index,Message:word):word;
index 38
procedure DisableAllNotifications; index 39
procedure ReEnableAllNotifications; index 40
procedure StartTerminalDownload(FilePath:pchar;ProtocolCode:integer); index 41
procedure StartTerminalUpload(FilePath:pchar;ProtocolCode:integer); index 42
procedure InterruptFileTransfer; index 43
procedure EstablishCommPortLink; index 44

procedure SupplyRegistrationCodes(Code1,Code2:pchar); index 45
(This last one is used by registered owners to remove shareware messages)

Messages Listing...

MESSAGE	COMMENT

1	Answering...	Server Only
2	Asking for last block	During File Transfers
3	Bad Block...	During File Transfer
4	Calling....	Client Only
5	Cancel Requested	
6	Checking COM....	During modem-location
7	Checking for required update files	Server Only
8	Client Aborted Move	Server Only
9	Client Does Not Have File	Server Only
10	Collecting Mail...	Server Only
11	Comm port not responding...	
12	Connected	
13	Connect Failed - Shutting Down...	Client Only
14	Connection Terminated	
15	Connection could not be established	Client Only
16	Copying...	
17	Could not create...	Uploads
18	Could not find modem	During Modem-Locate
19	Deleting...	
20	Denying Move...	
21	Download Complete	
22	Download Complete - UnZipping...	
23	Download Started	
24	Download request received	
25	EZDialup Server Reset...	
26	EZDialup Server Shutdown	
27	EZDialup Shutdown	
28	Error in opening file....	During an Upload
29	File Removed	After succesful Move
30	File Transfer Ended	
31	File-Data Block Acknowledged	During Ymodem Transfers
32	File-Data Block Requested	During Ymodem Transfers
33	Finding mail to send...	Server Only
34	Found modem at ...	During Modem-Locate
35	Good Block ...	During File Transfers
36	Hanging Up	
37	In Sync - sending reply	Server Only
38	Incorrect Checksum...	During File Transfers
39	Initializing Modem...	
40	Line Busy - Shutting Down....	Client Only
41	Locating Modem...	
42	Modem Not Initialized	
43	Modem Ready	
44	No Dialtone - Shutting Down....	Client Only
45	Other Size Zipping...	
46	Placing call...	Client Only
47	Protocol Established	
48	Ready to accept command	Mode C only
49	Received Block...	During File Transfers
50	Received Deny ACK	During File Moves
51	Received Remote Command	Server Only, Mode C
52	Received data block...	During Ymodem Transfers
53	Required update files loaded...	Server Only
54	Resending EOT...	During File Transers
55	Running program...	When either side runs program
56	Sent Command To Server	Client Only, Mode D

57 Sent Download Request
58 Sent EOT
59 Sent Init1 to modem
60 Sent Init2 to modem
61 Server Denied Move **Client Only**
62 Server Does Not Have File **Client Only**
63 Starting Download
64 Syncronized... **Mode C only**
65 This session did not end normally **Client Only**
66 Timed-asking last block... **During File Transfers**
67 Too many errors.... **During File Transfers**
68 Transfer Ended **During File Transfers**
69 UnCompressing Mail... **Server Only**
70 UnZipping...
71 Unable To Log In **Client Only**
72 Update requires more time... **Client Only**
73 Upload Complete
74 Upload Pending
75 Upload Started
76 Upload request received
77 Uploading...
78 Waiting For Mail **Client Only**
79 Waiting for Update **Client Only**
80 Waiting for call... **Server Only**
81 Will Download ...
82 Your information already current **Client Only**
83 Zipping Complete
84 Zipping...

Legal Stuff...

Disclaimer: Neither **EZ Software**, nor its officers, shall be responsible for any loss resulting from the use of **EZDialup.**

The shareware distribution file *EZDIALUP.ZIP* may be copied freely and used without restriction by anyone on any number of machines. However, this software remains the intellectual property of **EZ Software**, and use of this software in its free, evaluation version *for more than 60 days* is prohibited. All rights reserved - decompiling or reverse engineering of this software is prohibited.

The registration fee grants your use of the software for an unlimited number of users, but you may not re-sell the software unless you are terminating your own use of it entirely, and in that case may do so only one time. Developers who use EZDialup to write communications programs may include all required files when their application is distributed.

Version Chronology

1.75	**5/1/96**	**Modem auto-detect; message numbers sent with string messages**
1.71	**3/26/96**	**Improved error-recovery on poor connections**
1.7	**2/27/96**	**Mode D added; automatic downsizing of blocks on slow connections**

1.6	**1/15/96**	**Mode C added**
1.5	**12/10/96**	**Allows program executions on client and server**
1.4	**11/29/95**	**.LNK files renamed to .EZD to accomodate Windows 95**
1.3	**11/20/95**	**Mode B added**
1.2	**10/27/95**	**Detects busy, no-dialtone, no-carrier; Server resets if no carrier**
1.1	**9/28/95**	**First release as shareware**
1.0	**2/1/95**	**Began live use/testing of system software**

Registration and Support

This shareware contains all features available in registered EZDialup, but, until the product is registered, the client-mode copies will remind users, *after* each call, to register. When testing, just hit the Enter key in response to the Registration box that pops up.

Please send any questions or comments to **102732.472@compuserve.com**. You can expect a prompt reply.

Use Compuserve's Software Registration Service (GO SWREG) to register the software, via your Compuserve account, for **$75**. The registration number is **7676**. Compuserve will then inform EZ Software via e-mail that you have registered, and we'll e-mail to you the registration codes promptly.

Or, e-mail us that the "check's in the mail" and send a check or money order for **$65** to:

EZ Software
P.O.Box 181302
Fairfield, OH 45018

If e-mail is not available to you, please include your name and an address to which we can mail the registration codes.