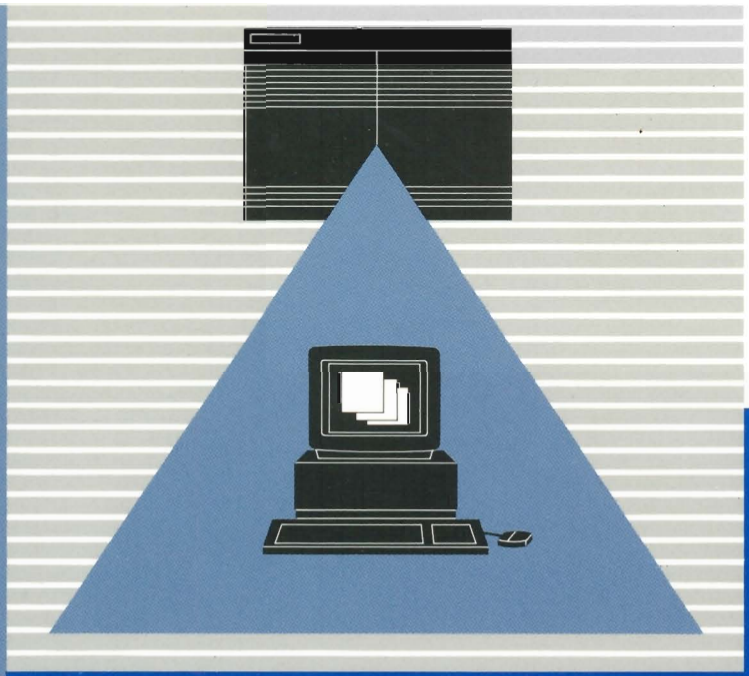
HEWLETT PACKARD

TermTalk

HP AdvanceLink's Script Language for MS-Windows® and the Apple Macintosh®



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

AdvanceLink's Script Language for
MS-Windows and the Apple Macintosh

TermTalk

HP Part No. 5960-2332
Printed in USA March 1991

First Edition
E0391

Notice

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on any particular item of equipment, unless that particular item of equipment upon which the software is used is being supported by Hewlett-Packard.

Preface

The TermTalk script language automates terminal-based tasks for users of AdvanceLink.

The information in this manual is organized as follows:

- Chapter 1, **Getting Started**, explains what you can do with TermTalk's scripting facility. It describes how scripts are created and used.
- Chapter 2, **Building TermTalk Scripts**, explains the various elements of the scripting language in detail.
- Chapter 3, **What the Commands Can Do**, groups the commands by function and provides some examples.
- Chapter 4, **Commands and Functions,** provides alphabetical listings of TermTalk commands, configuration settings, and functions, with syntax and explanation for each.
- Chapter 5, **Using Dynamic Data Exchange (DDE)**, describes how AdvanceLink supports Microsoft's DDE protocol for communications between applications. DDE-related commands and functions are described in detail. DDE is not available on the Macintosh platform.
- The **Appendices** include a list of system-defined terms, a table of equivalent commands in other command languages, and a discussion of host control. Following are the **Error Messages** and **Index**.

Contents

Getting Started 1

Introduction

This chapter describes the basic capabilities of a TermTalk script, and tells you how to use the commands on AdvanceLink's Script and Settings menus to create and run scripts.

What Scripts Can Do

A script is a sequence of statements written in TermTalk. When executed, a script automatically performs functions that would normally be performed by a user at a terminal. For example, a script can:

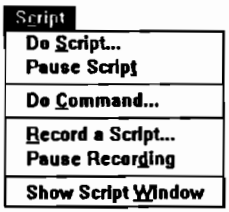
- Dial modems.
- Log on users.
- Set configuration values.
- Run applications.
- Transmit and log data.
- Create and delete files (on personal computer or host).
- Print files.
- Send and receive files to and from other computers.
- Prompt the user for input and receive input; display messages.
- Take different actions in different situations.
- Create and manage local screens; provide a user interface for host or PC-based applications.
- Exchange data with other applications using Microsoft's Dynamic Data Exchange (DDE) facility.

Though TermTalk is capable of performing very sophisticated functions, new users can easily set up simple scripts to automate routine tasks.

How Scripts Are Created

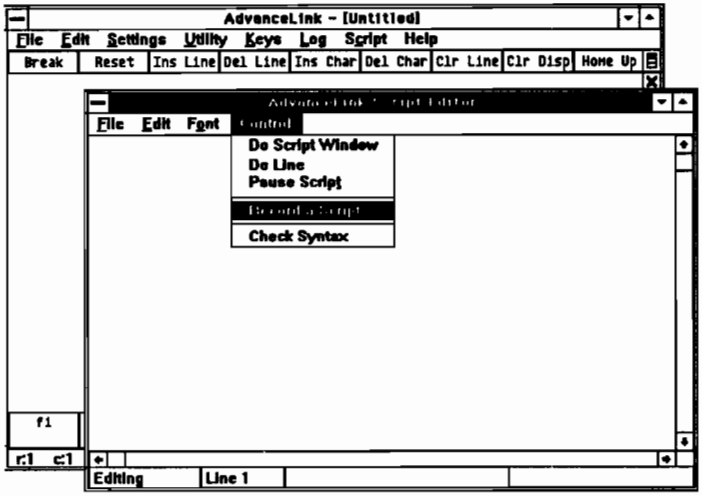
A TermTalk script can be created in one of three ways. You can:

- Turn on AdvanceLink's command recording facility while you perform a series of actions—the commands required to reproduce these actions are recorded automatically.
- Write and edit scripts in AdvanceLink's script window.
- Use any text file containing valid TermTalk commands.



You can turn on AdvanceLink's automatic command recorder by choosing **Record a Script** from the Script menu. You then perform the function you want to automate while AdvanceLink "watches" you and records the commands required to reproduce these functions to a script file. Or, if you would like to see the TermTalk commands displayed on the screen as you record them, you can choose **Show Script Window** to open the **script window**, then turn on recording by choosing **Record a Script** from the Control menu of the script window. Scripts created automatically can be edited if necessary, then saved and re-used time and again. More details follow in the section titled "Script Recording."

To write a script, choose **Show Script Window** from the Script menu. This opens the script window, which has its own menus with commands used to create and edit script files.



Also, any standard text file containing TermTalk commands can be run as a script.

If you have been using AdvanceLink for DOS or Reflection, you will need to convert your command files to TermTalk. The program `ADV2TTS.EXE` aids in the conversion of AdvanceLink scripts to TermTalk. The program `REF2TTS.EXE` does the same for Reflection scripts. See Appendix B for details. These programs run under DOS only (not Macintosh).

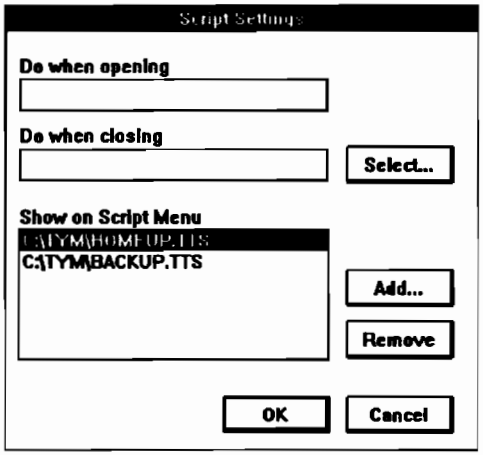
Executing Scripts

TermTalk scripts can be performed in several ways. You can

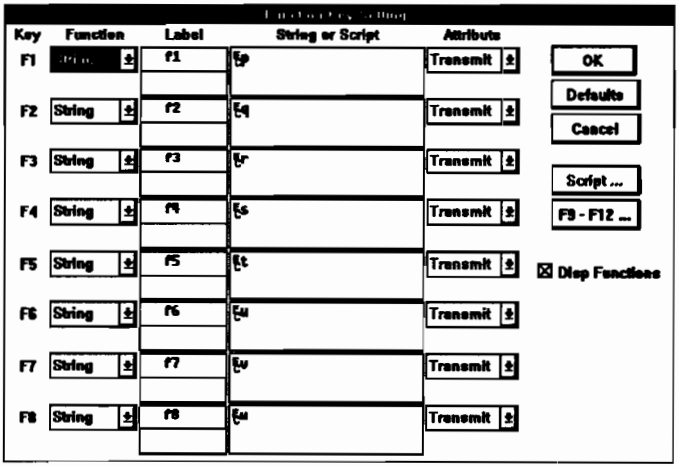
- Choose **Do Script** from AdvanceLink's Script menu, then choose the desired script in a dialog box.
- Place the names of frequently used scripts on the Script menu, so you can choose them directly.
- Associate the names of frequently used scripts with function keys, so they are executed when you press or click that key.
- Designate scripts to be performed automatically when you open and close your session file.
- Execute one command at a time by choosing **Do Command** from the Script menu.
- Execute the current contents of the script window, or any portion of it, using commands from the Control menu.
- Write a script that executes another script.

To perform a script which has been previously written and saved in a file, you choose **Do Script** from the Script menu. This opens a dialog box from which you can choose a script to execute. While the script is running, you can use the **Stop Script** and **Pause Script** commands to interrupt the script.

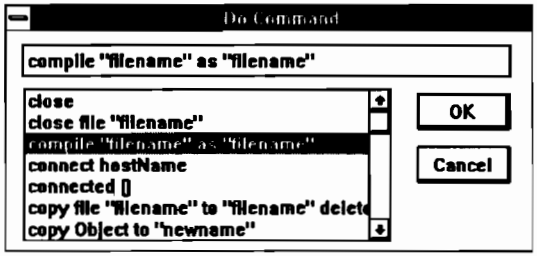
You can place the names of frequently used scripts on the Script menu so they can be run with one click of the mouse. Each session file can have a different list of scripts on the menu. To select the scripts to be displayed on the menu, choose **Scripts** from the Settings menu. The dialog box shown below is displayed. In this dialog box you can also specify a script to be executed automatically as the session file is opened, and another to be executed as it is closed.



To associate a script with a function key, choose the **Function Keys** command from the Settings menu. You can associate a script with a particular function key by selecting "Script" in the Function column, then clicking the **Script** button to select a particular script file. Like other user-created function key definitions, function key scripts are removed when a host application loads the function key values and labels.



When you choose **Do Command** from the Script menu, a dialog box is displayed in which you can enter and execute one command at a time. This facility can be used to perform simple actions or to test commands which are to be placed in a script.

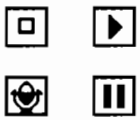


When a script is open in the script window, you can execute single lines, selected portions, or the whole of that script using the Control menu commands **Do Line/Do Selection** and **Do Script Window.**

A script can also be executed by another script, by placing the `do script` command in a script file. Scripts can be **nested** in this fashion up to fifteen levels deep.

Each host session you run can execute only one script at a time. If you are running multiple sessions, they can all execute scripts simultaneously. When a script is operating within a particular session (called **focusing** on that session), the user is locked out of the session until the script terminates or changes its focus to another session.

There is an indicator in the lower left hand corner of the main AdvanceLink window which shows when a script is being executed. This indicator has four states, as shown below.



No script activity: Hollow box.
Executing a script: Red arrow.
Recording a script: Red microphone.
Execution or recording paused: Red parallel lines.

Compiling Scripts

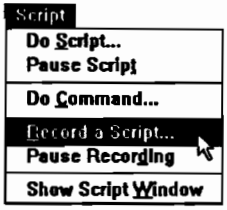
Like source programs written in general programming languages, TermTalk scripts must be compiled into object format before they are executed. Normally, you don't have to concern yourself with compilation as it is performed automatically the first time a script is run.

However, to streamline memory usage and to eliminate the possibility of an unforeseen compile-time error, you may wish to compile scripts which will be called from other scripts before they are actually used in a production environment. To do so, use the TermTalk `compile` command. Syntax for this command is found in the alphabetical command reference in Chapter 4. Note that simply saving a script with the **Save** or **Save As** commands from the File menu of the script window does not compile it.

File Extensions

Under Microsoft Windows, a source script file has the extension `TTS`. A compiled file has the extension `TTX`.

Script Recording



You can turn on AdvanceLink's automatic command recorder by choosing **Record a Script** from the Script menu. (This command changes to **Stop Recording** after it is selected. Both **Record a Script** and **Stop Recording** have the same accelerator, Alt + S + R, which allows you toggle easily between them.) Once recording is on, you perform the function you want to automate while AdvanceLink "watches" you. The TermTalk statements needed to reproduce these functions are recorded in a file.

When you choose **Stop Recording**, a dialog box is displayed in which you can name and save your script file. Under Microsoft Windows, the default extension for script source files is `TTS`.

To suspend recording temporarily while you are creating a script (for example, if you need to access another PC application), choose **Pause Recording** from the Script menu. This command then changes to **Resume Recording**. When recording is paused, the indicator in the lower left of the screen says "Paused". When you resume recording, commands are appended to the end of the file, where you left off.

To view and edit a script file created in this way, choose **Show Script Window** from the Script menu and choose **Open** from the script window's File menu to display the script.

Recording into the Script Window

To record a script directly into the script window, so that commands are displayed on the screen as they are recorded, choose **Show Script Window** from the Script menu, then start recording by choosing **Record a Script** from the Control menu of the script window. Do not select **Record a Script** from the Script menu of the main window, as this records commands to a file instead of into the script window.

When the script window is open and recording is in progress, **Record a Script** changes to **Stop Recording**. You can suspend or stop recording at any time by choosing this command. You can then save the file, discard the existing data, reposition the cursor, open a new file, and so on. If you choose **Record a Script** again, recording resumes at the current cursor position.

You can edit the script in the window at any time, and save it using the **Save** or **Save As** commands from the File menu. If you attempt to close the window without first saving the script, a dialog box is displayed in which you can name and save the file or discard it.

Recording Steps

To create a script that logs on to the HP 3000 and then performs a Home Up, Clear Screen, you could follow these steps:

1. Choose **Record A Script** from the Script menu.

2. Choose **Terminal** and/or **Connection** from the Settings menu to configure any communications parameters you need. Then type your logon and click the Home Up and Clear Display buttons.

3. Choose **Stop Recording** from the Script menu. Save your script by specifying a name in the dialog box displayed.

4. Log off, then test your script by choosing **Do Script** from the Script menu. If it does not work as you expected, you

can either open the script window to take a look at the script, or you can simply repeat the whole process in case you made a mistake the first time.

5. If you wish to have this script executed automatically when you open your session file, you would choose the **Script** command from the Settings menu of the main AdvanceLink window. You could also choose to have your logon script placed on the Script menu by making the appropriate settings in this box.

To record the same script with the script window open, you might follow these steps.

1. Choose **Show Script Window** from the Script menu, and then choose **Record a Script**. Then click in the main AdvanceLink window so that it becomes the active window again. You may wish to size the two windows so they are both visible on your screen.

2. Do the operation you want to record as described above. Commands appear in the script window as you perform each action.

3. Test the script by choosing **Do Script Window** from the Control menu. Edit the script as desired.

4. Choose **Save As** from the File menu, and type a file name for your script.

There are some things to look out for when using automatic command recording. For example, if you iconize AdvanceLink's window (under Microsoft Windows or NewWave), the window becomes an icon, and you will have to re-open it to continue creating your script. This gets recorded too, whether you like it or not. You can edit the script later to remove any unneeded commands.

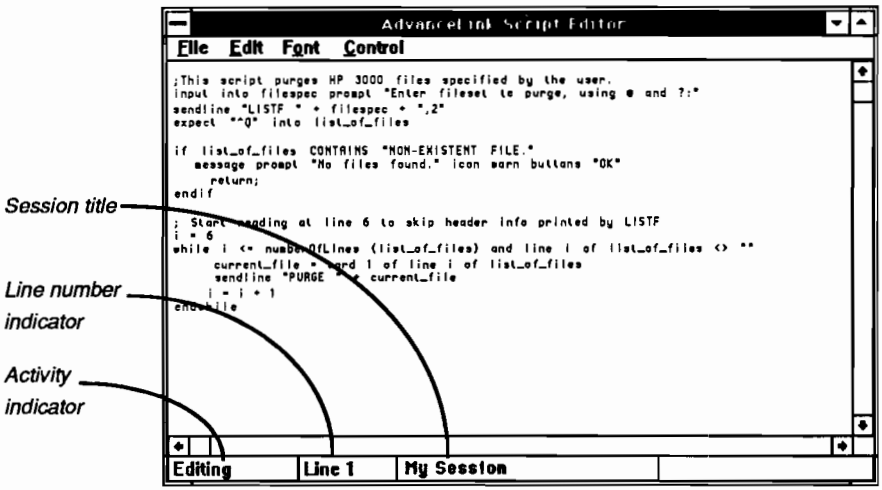
Also, if you record a script which performs complex interactions with host software, this script will very likely

require editing to fine-tune timing and handshaking procedures. You will probably need to use the TermTalk `expect` command to condition the sending of data upon the reception of the appropriate prompt from the host.

Using the Script Window

AdvanceLink's script window is a built-in text editor with which you can open, create, edit, and save scripts. To open the window, choose **Show Script Window** from AdvanceLink's Script menu. To close the window, choose **Hide Script Window.**

The title bar of the script window says Script Editor — Untitled (or the name of the script file).



The lower left corner of the window contains three indicators. The first shows the current activity. "Compiling" means a script is being translated into object format. "Running" means a script is being executed. "Paused" means that script

execution has been suspended. "Recording" means that command recording is in progress. "Editing" means that you are neither recording nor executing a script, and can add to or edit the contents of the window as you like.

The second indicator displays the current line number.

The third indicator displays the title of the focused session (the session from which you opened this window). The title is configured in the Terminal Settings dialog box (choose **Terminal** from the Settings menu of the main AdvanceLink window).

The window has four menus: File, Edit, Font, and Control. The commands on these menus are described below.

File Menu

The File menu (or Action menu, under NewWave) contains commands that manipulate script files.

New

Closes current file; creates and opens a new one. Not available under NewWave.

Open

Presents a dialog from which you can select an AdvanceLink script file or text file to be opened. On the Macintosh, the current file is closed and the one you select is opened. Under Windows, the file is opened into the current window and any existing contents are deleted. Not available under NewWave.

Save

Saves the contents of the current window to the file named in the title bar. If the window still has the default name, a dialog box is presented in which you can enter a different file name and select the drive on which the file is to be located. Under NewWave, this command updates the saved version of the TermTalk object.

Save As

Saves the contents of the current window, presenting a dialog box in which you can enter a file name and select a disk drive. Under NewWave, this command saves the object under a new name and places it in the NewWave Office.

Page Setup

Macintosh only. Presents a dialog box in which you can specify paper size, orientation, printing effects, and so on.

Attributes

NewWave only. Displays standard NewWave object attributes dialog box.

Printer Setup

Windows and NewWave only. Presents a dialog box in which you can specify paper size, orientation, and so on.

Print

Prints the contents of the window. On the Macintosh, the printer selected with the Chooser receives output. Under Windows, the printer designated in the Control Panel receives output.

Exit

Under Windows, closes the script window.

Close

Under Macintosh and NewWave, closes the script window.

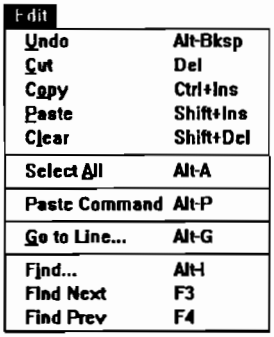
Convert

NewWave only. Imports TermTalk scripts from DOS to NewWave, and exports TermTalk objects to DOS.

About Script Editor

NewWave only. Provides version number.

Edit Menu



The Edit menu contains commands used to manipulate text in the script window.

Undo

Undoes the effect of the immediately previous insertion or deletion of text.

Cut

Removes selected text from the window and places it on the Clipboard.

Copy

Copies selected text, leaving it in the window, and places it on the Clipboard.

Paste

Places the text on the Clipboard in the window, starting at the current cursor location.

Clear

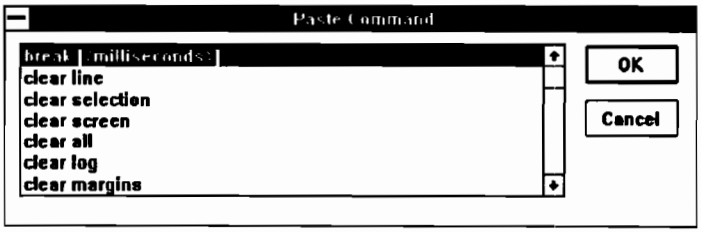
Removes the selected text from the window.

Select All

Selects all text in the window.

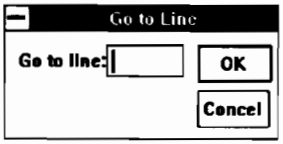
Paste Command

Presents a dialog box in which you can choose a command from a list of TermTalk commands. The selected command is pasted into the script window with its default syntax, where it can be modified as desired.



Go to Line

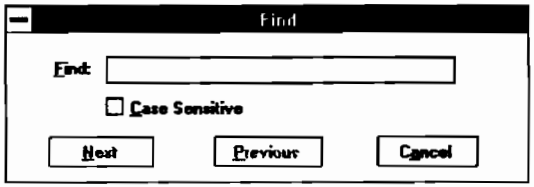
Presents a dialog in which you can enter a line number. When you click **OK**, the window is scrolled so the selected line is at the top of the screen, with the cursor positioned before the first character.



Find

Presents a dialog in which you can enter a string. When you click **Find Next** or **Find Prev**, AdvanceLink searches through the script window for a matching string, beginning at the point where the cursor is located. **Find Next** searches forward through the window; **Find Prev** searches backward.

Normally, a matching string need not be in the same combination of upper and lower case letters. To restrict the search to a string of identical case, choose the "Case Sensitive" option.



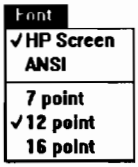
Find Next

Searches forward through the script window for the next occurrence of the last string specified in the **Find** dialog.

Find Prev

Searches backward through the script window for the previous occurrence of the last string specified in the **Find** dialog.

Font Menu



The Font menu contains commands that allow you to change the font and size of text in the script window.

HP Screen

The screen font of the AdvanceLink application. Includes Roman-8 international characters.

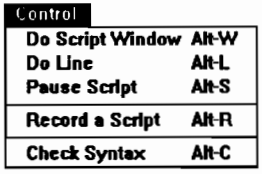
ANSI

The standard ANSI character set.

7, 12, 16

The default font size is 12.

Control Menu



The Control menu contains commands useful in creating and debugging scripts.

Do Script Window / Stop Script

Executes the entire contents of the script window. While the script is executing, this command reads **Stop Script**, and all other menu commands are disabled except **Pause Script.**

Do Line or Do Selection / Stop Script

Executes the line in which the cursor is currently placed. If text has been selected, this command reads **Do Selection**, and choosing it executes the highlighted portion of the window. While the line or selection is executing, this command reads **Stop Script**, and all other commands are disabled except **Pause Script.**

Pause Script / Resume Script

Pauses execution of the script initiated with **Do Script Window, Do Line,** or **Do Selection.** Once you have selected **Pause Script**, the command changes to **Resume Script.** Choose **Resume Script** to continue execution.

Record a Script/Stop Recording

Monitors actions performed by the user and places TermTalk commands which reproduce those actions in the script window. Once recording is in progress, this command changes to **Stop Recording.**

Check Syntax

Checks the syntax of the commands in the script window. If a portion of the text has been selected, the highlighted text will be checked, rather than the entire window. If an error is found, AdvanceLink displays a dialog box showing the line number and the error message. This command finds only one error at a time, so after correcting the first error located, you should choose **Check Syntax** again.

Building TermTalk Scripts 2

Introduction

This section describes the basic elements of the TermTalk script language, and explains the rules which govern their use. To illustrate most of the elements of the language, statements from the following sample script are used.

```
;This script purges HP 3000 files specified by the user.
input into filespec prompt "Enter fileset to purge, using @ and ?:"
sendline "LISTF " + filespec + ",2"
expect "^Q" into list_of_files

if list_of_files CONTAINS "NON-EXISTENT FILE."
  message prompt "No files found." icon warn buttons "OK"
    return;
endif

; Start reading at line 6 to skip header info printed by LISTF
i = 6
while i <= numberOfLines (list_of_files) and line i of list_of_files <> ""
    current_file = word 1 of line i of list_of_files
    sendline "PURGE " + current_file
    i = i + 1
endwhile
```

This script prompts a user to specify a fileset to be purged on the HP 3000. After the user's input is received, a `LISTF` command containing the file specification is sent to the HP 3000. If the files specified do not exist, an error message is displayed in a dialog box. If the files do exist, the names of the individual files are extracted from the listing produced by the `LISTF` command, and these files are purged one by one.

Sections of the preceding script are reproduced in the pages that follow to illustrate various aspects of AdvanceLink's script language. Boldface type is used to draw attention to the particular element under discussion.

Commands

At the heart of TermTalk are approximately 85 commands, which perform the following kinds of tasks.

Operation	Examples
User Interaction	`beep`, `input`, `message`
Configuration	`set`, `get`, `configure`, `hide`
Host Interaction	`break`, `delete char`
File transfer	`send`, `receive`
Working with Files	`open file`, `read`, `save`
Controlling AdvanceLink	`open application`, `quit`
Editing Data on Screen	`clear`, `cut`, `copy`, `paste`
HP NewWave	`send object`

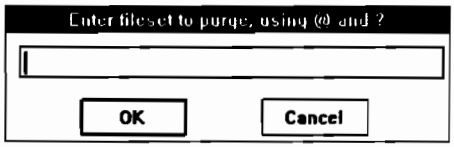
Each TermTalk command is described in the alphabetical reference in Chapter 4.

Commands can be continued over multiple lines by placing the ampersand character (&) at the end of each continued line.

In the sample script shown at the beginning of this chapter, four commands are used: `input`, `sendline`, `expect`, and `message`.

```
input into filespec prompt "Enter fileset to purge, using @ and ?:"
```

The `input` command displays a dialog box in which the user is prompted to enter a fileset specification:



The user's input is stored in the variable `filespec`.

```
sendline "LISTF " + filespec + ",2"
expect "^Q" into list_of_files
```

This `sendline` command incorporates the user's input into a `LISTF,2` command which is sent to the host. The `expect` command places the host's output into the variable `list_of_files` until the specified terminator, `^Q`, is received. `^Q` (DC1) is the HP 3000 prompt character.

```
message prompt "No files found." icon warn buttons "OK"
```

The `message` command displays a dialog box containing a user message. A warning icon is used to identify the message as an error condition, and an OK button is provided.

A second `sendline` command builds an MPE `PURGE` command to delete each specified file, and sends it to the host.

```
sendline "PURGE " + current_file
```

Variables

Variables play the same role in TermTalk scripts as in other programming languages—they are placeholders set up to contain information, either strings of text or numbers. You establish variable names as shown in the `input` command from the sample script at the beginning of this chapter.

```
input into filespec prompt "Enter fileset to purge, using @ and ?:"
```

`filespec` is a user-defined variable into which the file specifications entered by the user will be placed. In the command which follow `input` in the example,

```
sendline "LISTF " + filespec + ",2"
expect "^Q" into list_of_files
```

the contents of `filespec` are incorporated into the `LISTF` command string sent to the HP host. `list_of_files` is a variable as well, one which will receive the output of the file directory listing returned by the HP host. (Note that the + sign is used to assemble portions of a text string. This is called concatenation, and is discussed in detail in the section titled "Strings".)

In the preceding examples, variables are specified as part of a command. The sample script also includes variables which

are set up and used independently, rather than as part of a command:

```
i = 6
while i <= numberOfLines (list_of_files) and line i of list_of_files <> ""
current_file = word 1 of line i of list_of_files
sendline "PURGE " + current_file
i = i + 1
endwhile
```

Here, the variable `i` is used to track progress through the files listed in `list_of_files`. Also, the variable `current_file` is used to store the name of the file being purged.

Variable Names

Variable names can include upper and lower case letters as well as numbers and the underscore character, and may be up to 31 characters in length. The first character must be a letter or an underscore. A total of up to 255 characters may be used without causing an error but only the first 31 compose the unique name which identifies the variable.

If the variable has the same name as a TermTalk command, function, configuration setting, or other system-defined term, it must be terminated with a # symbol to distinguish it from that term. For example, if you wish to use `input` as a variable, you must specify `input#`, because `input` is a command. If you like, you can terminate all your variables with #, in HP NewWave agent style. This also prevents any possibility of conflict with TermTalk's many system-defined terms. For a complete list of system-defined terms, see Appendix A.

In addition to avoiding conflicts with system-defined terms, you must also avoid giving a variable the same name as other user-defined terms. These include user defined procedure names established with the `proc` statement, and go-to section names established with the `label` statement. See "User

Defined Procedures" and "Logic and Flow Control" later in this chapter.

Permanent Variables

Normally, the value of a particular variable is stored until the end of the script in which it occurs. If a variable with the same name is used in a different script, it will not pick up any value assigned to a variable with the same name in a previous script. You can, however, establish a permanent variable which remains defined throughout your entire session, and can be accessed by a number of different scripts. To do so, you use the `permanent` statement, as shown below:

```
permanent variable_name
```

Declarations of permanent variables must be the first statements in a script, preceded only by comments. The following two scripts illustrate the use of permanent variables.

```
;Script 1
permanent variable1, variable2          ;Declare permanent variables

variable1 = "New York"
variable2 = "Los Angeles"

display "The values at the beginning of Script 1 are:^M^J"
display variable1 + CR + LF             ; Show the values of the
display variable2 + CR + LF + CR + LF   ; variables

do script "Script2"                     ; Execute Script2 and continue

display "The values at the end of Script 1 are:^M^J"
display variable1 + CR + LF             ; Show the values of the
display variable2 + CR + LF             ; variables again
```

```
;Script 2
permanent variable1                    ; Declare variable1 permanent

variable1 = "Berlin"
variable2 = "Munich"

display "The values in Script 2 are:^M^J"
display variable1 + CR + LF            ; Show the values of the
display variable2 + CR + LF + CR + LF  ; variables
```

If you were to run Script 1, the output would look like this:

```
The values at the beginning of Script 1 are:
New York
Los Angeles

The values in Script 2 are:
Berlin
Munich

The values at the end of Script 1 are:
Berlin
Los Angeles
```

Because `variable1` is declared as permanent in both Script 1 and Script 2, it retains the value assigned to it in Script 2 even after Script 1 has resumed execution. `variable2`, however, is not declared as permanent in both scripts. Therefore the value it is assigned in Script 2 is not carried over when Script 1 continues.

Comments

Descriptive remarks, called comments, may be added to your script file at any point. A comment begins with a semi-colon(;).

```
; This script purges HP 3000 files specified by the user.
```

You cannot continue comments over multiple lines by using the ampersand character, as the ampersand will be ignored. Each comment line must begin with its own semi-colon.

```
; Comments are extremely important for making your script easy
; to read and understand.
```

Because AdvanceLink ignores any characters on an input line once it has encountered a semi-colon (unless the semi-colon is in a quoted string — see below), you can also place a comment following a short command or statement on the same line:

```
display variable1 + CR          ; show the values of the
display variable2 + CR          ; variables again
```

Note NewWave users should note that AdvanceLink's comment style differs from that of agent tasks, where apostrophe characters are used to introduce comment lines.

Strings

A string is any series of characters interpreted as text rather than numeric information. Most of the parameters of AdvanceLink's commands require string data. For example, the `message` command has the following syntax:

```
message [into variable] [prompt prompt_string]
[[icon] note|warn|stop] buttons string [,string
[,string]]
```

For each string parameter in the command above, you may specify either a literal string enclosed in quotes, or an expression that yields a string. In the example below, the strings "No files found" and "OK" are displayed exactly as shown in the command line.

```
message prompt "No files found." icon warn buttons "OK"
```

Alternately, you could specify a variable which contains string data.

```
myprompt$ = "No files found."
message prompt myprompt$ icon warn buttons "OK"
```

This too displays "No files found." Another way of specifying a string parameter is to build an expression that yields a string. For example, concatenated string expressions and chunking expressions (discussed below) both produce string data; so do some system-defined functions such as `getfilename()`.

You cannot pass numeric data to a command parameter which requires string data, or the converse. Use the `stringToNumber` and `numberToString` functions to perform data type conversions.

String Concatenation

The + symbol can be used to assemble, or concatenate, a number of string elements. For example, `"hello" + cr + lf` follows the string `hello` with a carriage return and a line feed. String concatenation is also used in the sample script :

```
sendline "LISTF " + filespec + ",2"
```

Here, an HP 3000 command string is built by putting together three elements:

- The string `LISTF`
- The contents of the variable `filespec`
- The string `,2`

Thus if `filespec` contained `@.PUB.PROGS`, the string produced would be

```
LISTF @.PUB.PROGS,2
```

The elements of a concatenated string expression can include:

- Quoted strings
- Variables that hold strings
- System-defined functions that produce strings
- Control characters

To place spaces between elements, include the spaces within the quoted strings.

Control Characters

In general, to send a control character from a TermTalk script, you use the standard alphanumeric control character abbreviations shown in the first column of the table below. However, when specifying a control character within a quoted string, you must take a different approach. Because the abbreviation is transmitted literally rather than acted upon if placed within quotes, a caret plus the appropriate letter or symbol should be specified instead.

outside quotes	inside quotes	decimal value	outside quotes	inside quotes	decimal value
NUL	^@	0	DLE	^P	16
SOH	^A	1	DC1	^Q	17
STX	^B	2	DC2	^R	18
ETX	^C	3	DC3	^S	19
EOT	^D	4	DC4	^T	20
ENQ	^E	5	NAK	^U	21
ACK	^F	6	SYN	^V	22
BEL	^G	7	ETB	^W	23
BS	^H	8	CAN	^X	24
HT	^I	9	EM	^Y	25
LF	^J	10	SUB	^Z	26
VT	^K	11	EC	^[	27
FF	^L	12	FS	^\	28
CR	^M	13	GS	^]	29
SO	^N	14	RS	^^	30
SI	^O	15	US	^_	31

For example, here are two ways to specify the word "hello" followed by a carriage return and a line feed.

```
"hello" + cr + lf
```

or

```
"hello^M^J"
```

When specifying control characters inside or outside quotes, case is not significant. Upper or lower case letters can be used.

Special Characters in Quoted Strings

Both the caret symbol and the double quote symbol have special meanings when used in quoted strings—^ signifies a control character and " terminates the string. Thus there is a special way to enter these characters within a quoted string so they are transmitted literally. Precede them with a tilde, as follows:

~" ~^

Because of this convention, if you created a quoted string with a tilde as the last character, the closing quotes would be transmitted literally rather than interpreted as terminating the quoted text. Therefore, to specify a literal tilde at the end of a quoted string, use two tildes in a row:

~~

For example, "abcd~~" produces the string abcd~.

A semicolon used in a quoted string is transmitted literally; it is not interpreted as the beginning of a comment.

Since a quoted string cannot span multiple lines, its maximum length is the maximum number of characters in an input line allowed by the text editor you use to create the script file, up to a limit of 255 characters.

Chunking

Chunking is a way of specifying part of a string to be extracted from the whole—similar to what is called "substringing" in some other computer languages.

For example, chunking is used in this portion of the sample script to extract the file names from the directory listing produced by the previous `LISTF,2` command, now stored in the variable `list_of_files`. `LISTF,2` produces a listing that includes one line per file. The first word of each line is the file name itself. Each file name is placed in the variable `current_file`, and sent to the HP 3000 to be purged.

```
i = 6
while i <= numberOfLines (list_of_files) and line i of list_of_files <> ""
   current_file = word 1 of line i of list_of_files
   sendline "PURGE " + current_file
   i = i + 1
endwhile
```

In the preceding example, the chunking keywords "word" and "line" are used to extract the desired portion of the listing.

There are four chunking keywords available:

Keyword	Meaning
`character`	Any ASCII character (abbreviated `char`)
`word`	A group of contiguous characters, including punctuation, delimited by spaces and/or end of line characters
`item`	A group of contiguous characters, including punctuation and spaces, delimited by commas
`line`	A group of contiguous characters, including punctuation and spaces, delimited by end of line characters (or, if the last line of the text does not include an end of line character, the end of the string)

These keywords are assembled using `of`:

```
char 5 of word 3 of "A quoted string like this"
```

This statement refers to the character "n" in the word "string."

You can also specify a range of elements, using `to`:

```
char 2 to 5 of word 3 of line 11 of output_text
```

This statement refers to four characters in the third word of the eleventh line of the text stored in the variable `output_text`.

Functions Used with Strings

The following system-defined functions operate on strings. Complete syntax of TermTalk's system-defined functions is found in Chapter 4.

Function Name	Purpose
`find`	Searches for substring in source string
`lower`	Converts uppercase to lower
`numberOfChars`	Returns number of characters in string
`numberOfItems`	Returns number of items in string
`numberOfLines`	Returns number of lines in string
`numberOfWords`	Returns number of words in string
`numberToString`	Converts numeric expression to string
`stringToNumber`	Converts string to numeric expression
`upper`	Converts lowercase to upper

For example, in this section of the sample script, the `numberOfLines` function is used:

```
i = 6
while i <= numberOfLines (list_of_files) and line i of list_of_files <> ""
   current_file = word 1 of line i of list_of_files
   sendline "PURGE " + current_file
   i = i + 1
endwhile
```

The `numberOfLines` function is performed on the argument `list_of_files` to determine how many lines there are in the list of files produced by the `LISTF,2` command. The result is the total number of individual files that have to be purged. This information is used to control the duration of the operation which purges the files one by one.

A Word about Upper and Lower Case

All of TermTalk's system-defined terms—commands, logical operators, function names, configuration settings, and so on—can be entered in either upper or lower case. A complete list of these is found in Appendix A.

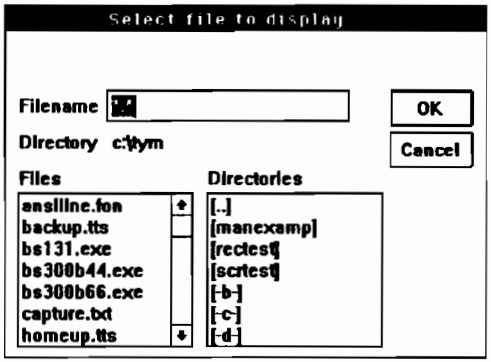
For quoted strings, case is significant. For example, if you compare the string "HELLO" to the string "hello," they are not considered equal. Lower case is evaluated as greater than upper case.

Empty String

"" (a pair of quotes with nothing between them) denotes an empty string. Testing for an empty string is useful to determine, for example, whether the user has entered data in response to a prompt.

```
; These commands display a dialog box in which the user enters a file name.
; If none is entered, the script terminates.
fileName# = getFileName("Select file to display:", "*.*")
if fileName# = ""
   stop
endif
```

A box like this is displayed:



If the user does not enter a file name, the variable contains an empty string. The `stop` command terminates the script.

Numeric Data and Arithmetic Expressions

Some of the parameters of AdvanceLink's commands require numeric data. These parameters are shown in italics in the command syntax provided in Chapter 4. For example:

```
tab [at column_number]
```

When numeric data is required, you must either supply a number or an arithmetic expression -- a sequence of operators and operands that computes a numeric value. For example, all of the following yield numeric values.

```
i + 1
baudrate/100
(5 + 3)/(x - 1)
5   (This is a numeric constant)
```

The following **arithmetic operators** can be used:

Operator	Meaning
+	Addition (also used to concatenate strings, as explained in the "Strings" section of this chapter)
-	Subtraction (if used with single operand, reverses the sign of the operand)
*	Multiplication
/	Division
MOD	Modulo (integral remainder from division)

The operands used in an arithmetic expression can be any of the following:

- Signed integers ranging from -2147483647 to 2147483647.
- Variables that hold numbers.
- System-defined functions that produce numbers
- Arithmetic expressions.

Note that floating point arithmetic is not available, and real numbers (numbers with fractional parts) are not supported.

Normally, expressions are evaluated from left to right. Multiplication, division, and modulo operations precede addition and subtraction. Parentheses can be used to control order of operations, with the expression in the innermost parentheses evaluated first.

The result of the evaluation of a particular arithmetic expression may be assigned to a variable with the = symbol. The variable is always specifed on the left side of the = sign, the expression on the right. For example:

```
i = i + 1
x = baudrate/100
value = (5 + 3)/(x - 1)
interest_rate = 5
```

You cannot pass numeric data to a command which requires string data, or the converse. Use the `stringToNumber` and `numberToString` functions to perform data type conversions.

Logic and Flow Control

TermTalk provides a number of special statements which control what parts of a script are performed and in what order. These keywords are shown in the sample script below.

```
label get_input
input into answer prompt "Enter Y or N: "

if answer = "Y"
  ; Do appropriate actions
elseif answer = "N"
  ; Do appropriate actions
else
   goto get_input
endif
```

In this script, the `input` command prompts the user to enter Y or N. If the user answers Y, one set of actions is performed. If the user enters N, an alternate set of actions is performed. If the user enters some response other than Y or N, the prompt is displayed again. To do this, the following keywords are used.

Keyword	Purpose
`if` `condition`	Sets condition. Followed by action to be taken.
`elseif` `condition`	Sets another condition. Followed by action to be taken.
`else`	Action to be taken if neither `if` nor `elseif` is true.
`endif`	Ends the current `if`.
`label` `label_name`	Specifies a name which identifies a particular part of the script, for use with `goto`.
`goto` `label_name`	Transfers processing to the part of the script identified by `label_name`.

Also useful in setting up logical conditions to control the flow of your script is the `while` keyword. In the sample script at the beginning of this chapter, `while` is used to establish a loop:

```
i = 6
while i <= numberOfLines (list_of_files) and line i of list_of_files <> ""
   current_file = word 1 of line i of list_of_files
   sendline "PURGE " + current_file
   i = i + 1
endwhile
```

Here the variable `i` is established to track progress through the lines of data contained in `list_of_files`. As long as `i` is less than or equal to the number of lines in `list_of_files`, and if the current line is not blank, the file on the line pointed to by `i` is purged. Once `i` is equal to the number of lines in `list_of_files`, the operation is complete.

Keyword	Purpose
`while`	Sets a statement or a series of statements to be executed repeatedly (a loop) until the controlling logical expression is false.
`endwhile`	Ends the current while.

The relational operators listed on the following page are useful in developing logical expressions. If two expressions are joined by the operator on the left, the resulting expression is true if the condition on the right exists. If not, the expression is false. False is defined to be the value zero; true is any non-zero value.

Operator	True if
<	First expression is less than second expression.
>	First expression is greater than second expression.
<=	First expression is less than or equal to second expression.
>=	First expression is greater than or equal to second expression.
<>	First expression is not equal to second expression.
=	First expression is equal to second expression.
`contains`	Second string is a substring of the first string.
`and`	First and second expression are both true.
`or`	First or second or both expressions are true.
`not`	The single expression to which it applies is false.

Note that you must compare numbers to numbers and strings to strings; you cannot compare arithmetic expressions to strings. (If you do, you will get the compiler error telling you that the operator you specified must be applied to operands of the same type.)

When comparing strings to strings, lower case is considered greater than upper case, and a longer string is considered greater than a shorter one.

The following additional commands can also be used to control the flow of a TermTalk script. These are fully documented in Chapter 4, "Commands and Functions".

Command	Purpose
`do`	Performs a user-defined procedure, established with `proc` statement.
`do script`	Performs the script in the specified file.
`return`	Terminates execution of current procedure or current script. If the current script was initiated by another script, returns control to initiating script.
`stop`	Terminates execution of all currently-active scripts.

System-Defined Functions

A function is a system-defined procedure that performs an operation and returns a value. The argument, or target of the function's operation, is always specified in parentheses. TermTalk's functions include:

Type of Function	Examples
String Functions	`find()`, `lower()`, `numberOfLines()`
Variable Functions	`isString()`, `isNumber()`, `isEmpty()`
System Functions	`freeDisk()`, `identity()`, `connected()`
File Functions	`newFileName()`, `getFileName()`
Error Handling Functions	`error()`, `errorLine()`, `errorString()`
DDE Functions	`ddeConversation()`, `ddeItem()`, `ddeMessage()`
HP NewWave Functions	`objectProperties()`, `agentTaskRunning()`

The syntax of each TermTalk function is described in detail in the alphabetical reference in Chapter 4.

In the following example, the `numberOfLines` function is used.

```
i = 6
while i <= numberOfLines (list_of_files) and line i of list_of_files <> ""
   current_file = word 1 of line i of list_of_files
   sendline "PURGE " + current_file
   i = i + 1
endwhile
```

The `numberOfLines` function is performed on the argument `list_of_files` to determine how many lines there are in the list of files produced by the `LISTF,2` command. This information is used to control the duration of the operation which purges the files one by one.

User-Defined Procedures

A user-defined procedure is a portion of the script that can be performed repeatedly without duplicating any code. The procedure is preceded and named by the `proc` statement and concluded with the `endproc` statement. To perform the procedure, use the `do` command and specify the procedure name established with `proc`.

The use of these statements is shown in the example below.

```
proc saveSettings
   get baud into oldBaud
   get hostPrompt into oldPrompt
endproc

proc restoreSettings
   set baud to oldBaud
   set hostPrompt to oldPrompt
endproc

do saveSettings
set baud to 9600
set hostPrompt to ":^Q"
sendline "hello manager.sys"
sendline "run myprog"
sendline "bye"
do restoreSettings
```

User-defined procedures must appear at the beginning of the script file, immediately following the declaration of any permanent variables.

Make sure the name you choose for your procedure does not conflict with any label name or variable name you are using in the same script, or with any system-defined term listed in Appendix A. A procedure name may be terminated with a # symbol to distinguish it from a similar label, variable, or system-defined term.

Error Handling

Two types of error can occur during execution of a TermTalk script: fatal and nonfatal. If a fatal error occurs during the execution of a script—for example, arithmetic overflow or division by zero—the error is reported and all currently-active scripts are terminated.

If a non-fatal error occurs, AdvanceLink's default is to report the error and terminate execution of all currently active scripts. However, you can override this default by specifying the `ignore errors` command. To return to default operation, specify `trap errors`. If scripts have been nested, `ignore errors` affects only the curently-executing script.

TermTalk also provides a number of functions which control error handling. These functions are only useful after an `ignore errors` statement, since otherwise errors cause termination of the current script. The functions include:

Function	Purpose
`error()`	Monitors error status of previous command: returns error number or zero for "no errors."
`errorline()`	Returns source file name and line number where last error occurred.
`errorstring (expression)`	Returns error message corresponding to the error number specified by the expression. To retrieve the error message for previous error, specify `errorstring(error())`.

The following scripts show some error handling techniques.

```
ignore errors                          ; Do not quit if an error occurs
open file "MYFILE"

if error()                             : If error occurs on open file
   create file "MYFILE"                ; file doesn't exist. Try to create it.
   if error()                          ; If this fails, show line # and msg.
      display errorline() + errorstring(error())
      stop
   endif
endif
trap errors                            ; Monitor errors again
```

```
ignore errors                          ; Do not quit if an error occurs
open file "MYFILE"

if error()                             : If error occurs on open file
   create file "MYFILE"                ; file doesn't exist. Try to create it.
   if error()=203                      ; Check for the error "FileExists"
   display "That file already exists."
   stop
   endif
endif
trap errors                            ; Monitor errors again
```

What the Commands Can Do 3

Introduction

This chapter groups TermTalk's scripting commands and system-defined functions in terms of the tasks they are used to perform. These tasks include:

- Communicating with the User
- Setting and Retrieving Configuration Values
- Communicating with the Host
- File Transfer
- Working with Files
- Controlling AdvanceLink
- Editing Data on the Screen
- Working with HP NewWave

In each section, a list of commands relating to the task is provided, along with an example illustrating how they are used. Most of these examples illustrate other commands and techniques as well. Going over them is a good way to familiarize yourself with TermTalk.

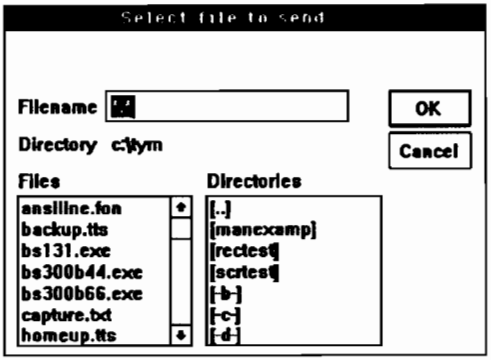
Refer to Chapter 4 for complete details on each command, along with additional examples.

Communicating with the User

These commands are used to exchange information with the person who is running the script.

Command	Purpose
`input`	Displays a dialog box in which user can enter a text string.
`message`	Displays a message box.
`display`	Displays a string or the result of an expression on the terminal screen at the current cursor position.
`beep`	Produces an audible tone.

Also useful are the functions `getFileName()` and `newFileName()`, which display dialog boxes in which the user can choose or create a file, respectively. The prompt you specify is displayed in the bar across the top of the box, as shown in this sample `getFileName` dialog.

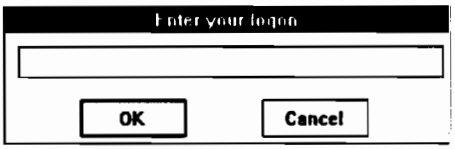


Example

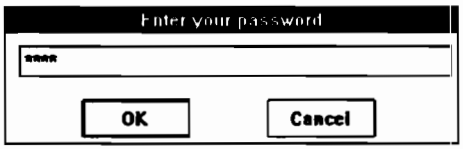
This example uses the `input`, `beep`, and `message` commands, as well as `sendline`. `input` is used to prompt the user for his logon and password. `beep` is used after logging on to get the user's attention, then a simple message is displayed with the `message` command.

```
; Logging on to the HP 3000
; Prompt the user for his logon string with the input command

; Test for empty string, returned if user clicks Cancel
input into logOnString prompt "Enter your logon:"
if logOnString = ""
   stop
endif

; Prompt the user for his password string with the input command
input password into passwordString prompt "Enter your password:"

; Test for empty string, returned if user clicks Cancel
if passwordString = ""
   stop
endif

; Send logon to the HP 3000 and wait for prompt
sendline logOnString
expect ":" + dc1
; Send the password to the HP 3000 and wait for prompt
sendline passwordString
expect ":" + dc1

; Tell the user he or she is logged on
beep
message "Logged on." buttons "OK"
```

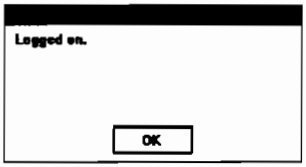
The first `input` command produces a dialog box like this:



The second `input` command produces a dialog box like this:



The message command produces a dialog box like this:



Setting and Retrieving Configuration Values

These commands are used to manipulate host configuration values.

Command	Purpose
`set`	Changes a configuration setting to the specified value.
`get`	Retrieves one of AdvanceLink's configuration settings into the specified variable.
`setup`	Creates a dialog box to allow the user to change AdvanceLink's configuration.
`open session`	Opens an AdvanceLink configuration file (a "session file") and sets the configuration options contained therein.
`save session`	Saves the current configuration to a session file.
`revert`	Returns all configuration settings to the values saved in the current session file, or if no session file is open, to AdvanceLink's default values.

Example

AdvanceLink's `set` and `get` commands are used to specify and retrieve the contents of a variety of system-defined configuration settings. For example, the setting `fontsize` controls the font used in the main AdvanceLink window. It can be set to `6`, `10`, `16`, or any other supported size. A complete list of configuration settings is found in Chapter 4.

The script on the next page specifies communications and terminal parameters as well as some screen display elements. Settings are then saved to a configuration file. In the section of the script labelled "Terminal Settings," notice that `yes`, `on`, and `true` can be used interchangeably when setting configuration parameters, as can `no`, `off`, and `false`.

```
; Configuring the Terminal and the Screen

; Communications settings
set the hostPrompt to ":" + dc1
set the baud to 9600
set the dataparityBits to eightNone

; Terminal settings
set display to yes
set smoothScroll to no
set enqAck to on
set typeAhead to off
set autoLineFeed to true
set blockMode to false

; Function key settings
set f1 label to "log on"
set f1 message to "hello manager.sys"
set f8 label to "log off"
set f8 message to "bye"

; Screen settings
set the text window location to "30,40"
set the text window title to "HP 3000 Defaults"

; save all settings to the AdvanceLink Defaults file
save session "AdvanceLink Defaults"

; the session file could be opened like this
open session "AdvanceLink Defaults"

; Here, the baud setting is retrieved using the get command, set to a new rate
; then reset to the original value
get the baud into originalBaudRate#
set the baud to 1200
set the baud to originalBaudRate#
```

Communicating with the Host

These commands are used to interact with the host computer.

Command	Purpose
`break`	Sends a break signal to the host computer.
`connect`	Connects to a host computer.
`disconnect`	Terminates the connection to the current host computer.
`dial`	Dials a phone number through an attached modem.
`sendline`	Sends a string plus a carriage return to the host computer.
`sendstring`	Sends a string, without a carriage return, to the host computer.
`expect`	Receives data from the host computer.
`key`	Simulates pressing various keys on the terminal keyboard.

Examples

The following sample script connects to an HP 3000 using a LAN, reads the current jobs running on that system into a file, then disconnects. It uses the `connect`, `disconnect`, `sendline`, and `sendstring` commands to do so. It also demonstrates how to use AdvanceLink commands to create a file, open it, write to it, and close it.

```
; Communicating with the Host

; first ask the user for the name of the desired host
input into myHostName prompt "Enter the host for job listing:"
; Test for empty string, returned if user clicks Cancel
if myHostName = ""
   stop
endif

; now connect to the host specified by the user
connect myHostName

; log on to the host computer using another script called "logon"
do script "logon"

; issue the SHOWJOB command using sendline, then wait for prompt
; the job listing output from the host is stored into the variable jobList
sendline "showjob"
expect ":" + dc1 into jobList

; save the job listing in a file whose name is created from the host name
jobListFileName = char 1 to 8 of myHostName + ".JOB"
; the jobListFileName now looks like "series70.JOB"
; create and open the file
create file jobListFileName
; write the job listing obtained aboved to the file
write jobList to file jobListFileName
; and close the file
close file jobListFileName

; log off the host
; since sendstring doesn't append a carriage return for us, add one
; with by concatenating cr to the end with +.
sendstring "bye" + cr

; disconnect from the host
disconnect
```

The next script dials an HP 3000 using the modem, gets a prompt, and displays a message on the terminal screen. It uses the `dial`, `break`, and `key` commands, among others.

```
; Dialing In To A Host

; first dial the host computer using the modem
dial "555-1212"

; get the host's attention using break followed by carriage returns
break
key return 3 times
; make sure we have the host's attention
sendstring cr
expect ":" + dc1

; now display a message on the terminal screen
; first home up and clear screen
cursor homeUp
clear screen
display esc + "&dB" + "Log in to your host machine:" + esc + "&d@"
```

File Transfer

These commands are used to send and receive files with the host computer. Transfer protocol and file conversion options are specified as command parameters.

Command	Purpose
`send`	Sends a file to the host computer.
`receive`	Receives a file from the host computer.

Example

This sample script downloads a file on the host which contains a list of files to be backed up. It then transfers each of the files listed.

```
; Backing Up Files To the Host

; first, wait until 1 AM to start the backup
wait until "01:00"

; receive a file "download" from the host to the PC file "files.txt" as a text
; file, deleting it if it already exists
receive "download" to "files.txt" as text delete

; open the file just received
open file "files.txt"

; read the first line of the file into the variable currentFile
read file "files.txt" for 1 line into currentFile#
; loop through the list until currentFile is an empty string,
; meaning we have reached the end of the file
while currentFile# <> ""
; send file in currentFile to host, deleting it if it already exists
send file currentFile# as backup delete
; now read the next file to send from the file
read file "files.txt" for 1 line into currentFile#
endwhile

; close the file list
close file "files.txt"
```

Working with Files

These commands are used to manipulate files on your PC or Macintosh.

Command	Purpose
`copy file`	Copies an existing disk file.
`delete file`	Deletes an existing disk file.
`list files`	Places a list of files into the supplied variable.
`rename file`	Renames a file on disk.
`close file`	Closes a file opened with `open file` or `create file`.
`create file`	Creates and opens a new ASCII text file.
`open file`	Opens an ASCII text file.
`print`	Prints a text file, the graphics screen, or the current log.
`read file`	Reads from the current read file pointer of a disk file previously opened with `open file` or `create file`.
`save`	Saves AdvanceLink data into a file.
`write`	Writes an expression to the write file pointer of a disk file previously opened with `open file` or `create file`.

Example

This sample script saves the current screen contents to a file and sends that file to the printer. This is a convenient way to print a copy of your screen without affecting any logging that may be in progress.

```
; Printing Data on the Screen

; First we copy the contents of the screen to a variable called 'theScreen'
select screen
copy
paste into theScreen

; Create 'tempfile', write the contents of 'theScreen' to it, and close it.
create file "tempfile"
write theScreen to file "tempfile"
close file "tempfile"

; Now print the file on the printer
print file "tempfile"

; Delete the temporary file
delete file "tempfile"
```

Controlling AdvanceLink

These commands are used to control various elements of AdvanceLink's operation.

Screen Control

Command	Purpose
`maximize`	Enlarges the specified window to full size of monitor.
`minimize`	Collapses AdvanceLink application to an icon.
`show`	Makes the specified display element visible to the user.
`hide`	Hides the specified display element from the user's view.
`restore`	Restores AdvanceLink's windows to their normal size.

Script Control

Command	Purpose
`compile`	Translates a source script into a compiled script.
`do`	Performs a user-defined procedure established with `proc` statement.
`do script`	Invokes the specified script.
`return`	Terminates execution of current procedure or current script. If the current script was initiated by another script, returns control to initiating script.
`stop`	Terminates execution of all currently-active scripts
`wait`	Pauses script execution until a specified time.

`ignore errors`	Causes AdvanceLink to ignore non-fatal errors, so that no message is displayed and the script continues execution.
`trap errors`	Reinstates error trapping, so that whenever an error is found, a message is displayed and the script terminates.

Miscellaneous

Command	Purpose
`open application`	Opens an application.
`page setup`	Displays a dialog box containing page setup options.
`reset`	Resets the terminal.
`quit`	Terminates execution of AdvanceLink.
`log`	Logs data in display memory. Logged data can be printed with `print log`, saved to a disk file with `save log`, copied to the Clipboard with `copy log`, or cleared with `clear log`.

Examples

This script sets the `logDirection` to top, so that lines are logged as they scroll out of display memory, then turns logging on. When logging is finished, the logged data is saved to a file.

```
; Logging Screen Data
set logDirection top
start logging

; To stop logging later on
stop logging
; To save the log file
save log "mylog"
```

This script automates routine tasks a user might run on a daily basis, freeing the user to take care of other business.

```
; Automating Routine Tasks

set transferProtocol to link3000
set hostPrompt to ":^Q"

do script "LOGON3000"                    ; Run logon script
do script "PRINTMAIL"                    ; Run script to print mail

ignore errors                            ; Do not quit if an error occurs
receive "BACKSTAT" delete                ; Transfer a file to PC
if error()
   result = "unsuccessful"
else
   result = "successful"
trap errors                              ; Restart error monitoring
sendline "BYE"                           ; Log off

; Display a message telling the user the job has finished
message prompt "Your daily routine is finished^M" +     &
       "Your mail has been sent to the printer^M" + buttons "OK"
quit
```

Editing Data on the Screen

These commands are used to modify data on the screen.

Terminal Editing

Command	Purpose
`delete char`	Deletes character under the cursor and moves rest of the line one space left.
`delete line`	Deletes line cursor is on and moves rest of display up one line.
`insert char`	Switches between insert character mode and overstrike character mode.
`insert line`	Inserts a line before the line the cursor is on and moves all lines below the new line down.
`cursor`	Controls cursor movement.
`graph`	Performs graphics functions.

PC Editing

Command	Purpose
`clear`	Removes specified element.
`copy`	Copies information to clipboard.
`cut`	Places information on the clipboard and deletes the original.
`paste`	Pastes the contents of the clipboard to display memory at the current cursor location or into the specified variable.
`select`	Selects or deselects parts of display memory.
`tab`	Sets a tab at the specified column.

Examples

This script allows the user to set a series of tab positions.

```
; Set a special group of tab settings:
MyTabSettings# = "7,8,12,16,20,24,28,32,36"
i = 1
while i <= numberOfItems(MyTabSettings#)
   thisTab# = stringToNumber(item i of MyTabSettings#)
   tab at thisTab#
   i = i + 1
endwhile
```

This script allows the user to select a group of lines on the screen, delete them, and replace them with the contents of the clipboard.

```
; This script replaces a range of lines with the contents of the clipboard.

input into deleteLines# prompt "Enter first and last line separated by a comma:"
& with "1,24"
; Check for Cancel or empty string
if deleteLines# <> ""
   firstLine# = stringToNumber(item 1 of deleteLines#)
   if numberOfItems(deleteLines#) = 2
      lastLine# = stringToNumber(item 2 of deleteLines#)
   select line firstLine# to lastLine#
else
   select line firstLine#
endif

clear selection                    ; Remove data from selected lines
cursor firstLine#                  ; Move cursor to first line
paste                              ; Insert data
endif
```

Working with HP NewWave

When a NewWave object is dragged and dropped or pasted from the clipboard into the AdvanceLink for NewWave window, it becomes the "current object" for the purpose of TermTalk's commands and functions. By default, the current object is transferred to the host as a Serialized Object File (SOF) package. To establish a different automatic action, or to customize or disable file transfer, you can designate a particular TermTalk object to be executed on object drop or paste. The commands listed below are designed to be included in such a script.

More information on autoexecution of TermTalk objects can be found in the manual titled *AdvanceLink NewWave Supplement.*

Commands

Command	Purpose
`export object`	Exports (or serializes) the current object to the named MS-DOS file.
`import object`	Imports a serialized object from MS-DOS.
`send object`	Transmits the current object to the host computer.
`receive object`	Receives a file from the host and imports it as an HP NewWave object.
`do script object`	Runs a TermTalk object.
`do agent task`	Runs a NewWave Agent task.
`set|get helpFile`	The name of the help file used by AdvanceLink for NewWave.

Functions

Function	Purpose
`currentObject()`	Returns TRUE if there is a current object available for processing.
`objectProperties()`	Returns a string supplying information about the properties of the current object.
`agentTaskRunning()`	Returns TRUE if an HP NewWave Agent Task is currently running.

Examples

The following script might be used to integrate NewWave with HP Desk, allowing objects dropped into the AdvanceLink for NewWave window to be mailed to other HP Desk users. This script assumes that HP Desk has been configured to be able to run the HPLink file transfer host program. This may be done either by creating an HP Desk script called `:RUN` or by changing AdvanceLink's file transfer initiation string to `:RUN HPLINK.PUB.SYS`. Select **File Transfer** from the Settings menu to do this.

```
; Check the label text of function key 8 to see if HP Desk
; is currently in an appropriate state to receive an object
get f8 label into labelText
if not (labelText contains "Other Areas")  and not (labelText contains "Signoff")
message into buttonPressed prompt "HP Desk unable to receive object" &
  icon warn buttons "OK"
return
endif
; Upload the object to the host computer ready to send as a message
send object to "tempfile" delete
; make sure the Tray prompt is displayed after the transfer completes
sendline ""
; If object is a Text Note, allow user to specify distribution list
if line 2 of objectProperties() = "Text Note"
    send the object SOF file as a message
   sendline "send (tempfile)"
   ; use object title as message subject
   sendline line 1 of objectProperties()
   ; The user can now be allowed to complete the distribution list.
   expect "MESSAGE > ^Q" wait 1 hour interactive
else
   ; If not Text Note, let user specify distribution list and explanatory text
   ; start to send a message
   sendline "send"
   ; use object title as message subject
   sendline line 1 of objectProperties()
   ; now wait for user to complete the distribution list and descriptive text
   expect "MESSAGE > ^Q" wait 1 hour interactive
   ; once at the message prompt import the object being sent into HP Desk
   sendline "copy from (tempfile)"
   ; again use the object's title as the item subject
   sendline line 1 of objectProperties()
endif
; Prompt user to confirm message then send.
message into buttonPressed prompt "OK to send?"  buttons "OK", "Cancel"
if buttonPressed = 1
   sendline "mail"
endif
```

The `do agent task` command allows TermTalk objects to start NewWave Agent tasks. One such task might automatically back up NewWave objects to a host computer. The following script logs on to the host, then runs an Agent task which drops each file to be backed up into the open AdvanceLink window. For a complete discussion of this subject, see Chapter 1 of the *AdvanceLink NewWave Supplement*.

```
; Get the host computer's attention
sendstring cr+cr
expect ":^Q"
; Now we have a colon prompt - logon
sendline "hello name.user/acctpass"
expect ":^Q"
; Now logged on - start Agent task to drop objects in the
; window. When all objects have been backed up the Agent
; Task will close the AdvanceLink window.
do agent task "Backup"
```

Note that TermTalk script execution always takes precedence over task execution. If an Agent task issues a command to an AdvanceLink object which is already running a TermTalk script, the Agent command is not executed until the TermTalk script has finished.

Commands and Functions 4

Commands

Using this Section

Please take note of the following when using the command syntax in this section.

Keywords

Keywords that must be entered exactly as shown are in boldface type. For example:

```
copy file source_file to target_file [delete]
```

User-Specified Parameters

Parameters for which you substitute the value of your choice are in regular type, like both `source_file` and `target_file` in the example above.

String and Numeric Data

Command parameters that require string data are shown in normal, non-italic type.

```
do script script_name
```

Parameters that require numeric data are in italics.

```
cursor row_number[,column_number] [relative]
```

When substituting values for command parameters, you may specify either an expression which evaluates to that value, or the value itself. If the value is a string literal, it should be enclosed in quotes. For example, the `do script` command has the syntax:

```
do script script_name
```

If you specify `"myscript"` in quotes, it is considered to be the literal name of the script file.

```
do script "myscript"
```

If you specify `myscript` without quotes, it is considered to be a variable containing the script name.

```
do script myscript
```

You could also specify an expression which would return a file name. For example,

```
do script getfilename("Which script?","*.*").
```

You cannot pass numeric data to a parameter which is expecting string data, or the converse. Use the `stringToNumber` and `numberToString` functions to perform data type conversions on the command line.

For example, in the script shown below, `stringToNumber` is used to convert string data to the numeric data required by the `tab` command.

```
; Set a special group of tab settings:
MyTabSettings# = "7,8,12,16,20,24,28,32,36"
i = 1
while i <= numberOfItems(MyTabSettings#)
   thisTab# = stringToNumber(item i of MyTabSettings#)
   tab at thisTab#
   i = i + 1
endwhile
```

Optional Items

Optional items are surrounded by brackets. For example:

```
[as id_number]
```

Choose One

A group of possible items from which you must choose one is separated by vertical slashes. For example:

```
command_name | script_name | string_to_send
```

On and Off

Note that `true`, `yes`, or any non-zero numeric value are equivalent to `on`. `False`, `no`, or 0 (zero) are equivalent to `off`. Thus `insert char on` is the same as `insert char yes`, or `insert char true`, or `insert character 1`.

Related Settings

The operation of some commands is affected by the setting of certain configuration parameters. For example, the `send` command is affected by the value of the `transferProtocol` configuration setting. Under each command where applicable, a list of related configuration settings is provided. The configuration settings themselves are described following the alpabetical command listing in this chapter.

Errors

For each command which returns error messages, a list of possible errors and error numbers is given. Error numbers can be tested with the `error()` function, as shown in the following example:

```
ignore errors                          ; Do not quit if an error occurs
open file "MYFILE"

if error()                             : If error occurs on open file
   create file "MYFILE"                ; tile doesn't exist. Try to create it.
   if error()=203                      ; Check for the error "FileExists"
   display "That file already exists."
   stop
   endif
endif
trap errors                            ; Monitor errors again
```

For More Information

In many cases, the operation of TermTalk commands which perform terminal emulation functions overlaps with that of AdvanceLink's menu and dialog box items. For example, the various forms of TermTalk's `log` command provide the same functionality as commands on AdvanceLink's Log menu. Although all of these terminal emulation functions are described in this manual, in some cases more detail can be found by looking up the corresponding menu command in the *AdvanceLink Reference* manual.

beep

Produces an audible tone.

beep

Related Settings `disableBell`

break

Sends a break signal to the host computer, temporarily interrupting PC/host communications.

break [*length_of_signal*]

Notes length_of _signal is specified in milliseconds; the maximum length is 5000 milliseconds.

```
; Send a 300 millisecond break on the connection
break 300
```

clear

Removes specified element.

clear line	Clears from cursor position to the end of the line.
clear selection	Clears current selection, if any.
clear screen	Clears text screen from cursor .position to end of screen.
clear all	Clears display memory.
clear graph	Clears graphics memory.
clear log	Clears current log.
clear margins	Clears left and right margin settings; equivalent to setting the left margin to 1 and setting the right margin to the value of the columns configuration setting.
clear tab	Clears tab set at the current column, if any.
clear all tabs	Clears all tab settings.
clear dde handlers	Cancels all existing DDE handler scripts and terminates all current DDE conversations. Available with Windows and NewWave versions only.

close file

Closes a file previously opened with `open file` or `create file`.

```
close file file_name
```

Notes

`file_name` must exactly match the file name specified with `open file` or `create file`.

```
; Open a file, write a message, and close the file
open file "TESTFILE.TXT"
write "My Message" to file "TESTFILE.TXT"
close file "TESTFILE.TXT"
```

Related Settings

`directory`

Errors

200 filePermissionError
201 fileNotFound
208 fileNotOpen

compile

Translates a source script into a compiled script.

```
compile source_script as compiled_script
```

Notes AdvanceLink recognizes the extension `.TTS` for source script files, and the extension `.TTX` for compiled script files.

```
; Compile a source file called mysource into the file called myobject
compile "mysource.tts" as "myobject.ttx"
```

Related Settings `directory`

Errors

200	filePermissionError
201	fileNotFound
202	fileIOError
203	fileExists
204	tooManyOpenFiles
205	fileBusy
206	fileTooLarge
207	diskFull
100	insufficientMemory

connect

Connects to a host computer.

connect [name_of_host]

Notes

connect is supported only in a LAN environment. When using a serial port this command is ignored.

name_of_host specifies the node name of the host computer. If name_of_host is not specified, connection is made to the host specified by the hostName configuration setting, which is initially hp3000, but may be changed with the set command.

```
; Connect to a host selected by the user
input into newHost$ prompt "Enter new host name:"
connect newHost$
```

Related Settings

baud
connection
hostName
modemType
phoneNumber
phoneType
port

Errors

250 connectFailed

copy

Copies information from a specified source to the clipboard.

```
copy [selection] [as table]
copy fields
copy graph [scaled]
copy windowImage
copy log
```

Notes

Specifying `copy` with no further parameters does the same thing as `copy selection`: it copies the current selection to the clipboard. If there is no current selection, the command does nothing. The keyword `selection` is only specified to improve the readability of your scripts.

Use the `select` command to mark a screen selection from a script.

On the Macintosh, a selection may be copied `as table`. `copy as table` is not supported under Windows.

On the Windows platform, `copy fields` copies all unprotected fields on the screen. `copy fields` is not supported on the Macintosh.

If `copy graph` is specified, the contents of the graphics window are copied to the clipboard. If `scaled` is specified, the contents are copied as scaled to fit in the window. `copy graph` is not supported with versions of AdvanceLink that do not support graphics.

If `copy windowImage` is specified, a bitmap of the current window is copied to the clipboard.

If `copy log` is specified, the contents of the current log are copied to the clipboard.

```
; Copy the current selection from the screen to the clipboard
copy
; Copy unprotected fields to the clipboard (Windows only)
copy fields
; Copy a bitmap of the current window to the clipboard
copy windowImage
```

Errors

100	insufficientMemory
200	filePermissionError (`copy log` only)
201	filenotFound (`copy log` only)
202	fileIOError (`copy log` only)
204	tooManyOpenFiles (`copy log` only)
205	fileBusy (`copy log` only)
391	errorAccessing Clipboard

copy file

Copies an existing disk file.

copy file source_file **to** target_file [**delete**]

Notes

source_file is the name of an existing file; the file need not have been opened with the open file command.

target_file may be any file name which is valid by the rules of the operating system under which AdvanceLink is running.

If target_file already exists and the delete keyword is specified, the file will be overwritten. If target_file exists and delete is not specified, a run time error occurs. If source_file does not exist or if the name supplied for target_file is invalid, a runtime error occurs.

```
; Copying files
; This script requests the two files names involved in the copy procedure.
; The new file will overwrite any existing file of the same name without
; any notification.
input into sourceFile# prompt "Enter source file name:"
input into targetFile# prompt "Enter target file name:"
copy file sourceFile# to targetFile# delete
```

Related Settings

directory

Errors

200	filePermissionError
201	fileNotFound
202	fileIOError
203	fileExists
204	tooManyOpenFiles
205	fileBusy
206	fileTooLarge
207	diskFull

create file

Creates and opens a new ASCII text file on the PC. Binary files are not supported.

```
create file file_name
```

Notes

A fully-qualified file name can be specified, with or without a file extension. If not otherwise specified, files are created in the directory where AdvanceLink is located, unless this default has been overriden by setting the `directory` configuration parameter or by changing the current directory in any Windows file selection dialog box.

Two file pointers are maintained for the file, one for reading and one for writing. Reading from a newly created file returns an empty string.

```
; Create a new file
create file "TESTFILE.TXT"
```

Related Settings

`directory`

Errors

200 filePermissionError
203 fileExists
204 tooManyOpenFiles
205 fileBusy
207 diskFull

cursor

Controls cursor movement and scrolling.

```
cursor row_number[,column_number] [relative]
cursor up
cursor down
cursor left
cursor right
cursor rollRight
cursor rollLeft
cursor rollUp
cursor rollDown
cursor homeDown
cursor homeUp
cursor prevPage
cursor nextPage
```

Notes

In the first form of the command the cursor is moved to the screen coordinates specified. These coordinates are specified in terms of the whole of display memory, not just the portion visible on the screen. In other words, 1,1 is the first row and column in display memory, not the first row and column on the screen.

If `column_number` is omitted, a default value of one is used.

If `relative` is specified, the cursor is moved the specified number of positions from the current location. Positive row and column numbers move down and to the right respectively; negative ones move up and to the left.

`cursor up`, `down`, `left` and `right` move the cursor one character in the specified direction.

`cursor rollRight`, `rollLeft`, `rollUp` and `rollDown` scroll the contents of the window one position in the specified direction. The commands are equivalent to the user clicking one of the arrows in the scroll bars one time.

`cursor homeDown` moves the cursor to the left margin of the line after the last line in display memory. `cursor homeUp`

moves the cursor to the left margin of the first line in memory.

`cursor prevPage` scrolls the display so that lines preceding the current screen in display memory, up to and including the line immediately before the top line, are displayed. The cursor is positioned to the top left position on the screen. For example, if the window is currently displaying 24 lines, the display is repainted so the previous 24 lines are displayed.

`cursor nextPage` scrolls the display so that lines following the current screen in display memory, beginning with the line immediately following the last line on the screen, are displayed. The cursor is positioned to the top left position on the screen. For example, if the window is currently displaying 24 lines, the display is repainted so the next 24 lines are displayed.

```
; Move the cursor around the screen.

cursor 20,10          ; Moves cursor to row 20, column 10

cursor 2,2 relative   ; Moves cursor 2 columns right and 2 rows down

cursor homeUp         ; Moves cursor to Home (1,1)

cursor right          ; Moves cursor one space to the right

cursor rollRight      ; Scrolls contents of window one space to the right

cursor prevPage       ; Scrolls contents of window up one "page" (24 lines)
```

Related Settings

`memoryLock`
`numberOfColumns`
`numberOfScreens`
`leftMargin`
`rightMargin`
`xmitFunctions`

cut

Copies information from a specified source to a specified destination and deletes the source.

```
cut [selection]
```

Notes

`selection` may be specified to improve the readability of your scripts; however, the command always copies the current selection. If there is no current selection, the command does nothing.

Use the `select` command to mark a screen selection from a script.

```
; Cut from selected area of screen to clipboard:
cut
```

Errors

391 errorAccessingClipboard
100 insufficientMemory

dde advise

Establishes a permanent link between AdvanceLink and the server application such that the server notifies AdvanceLink of the value of an item each time the value changes. Available with Windows and NewWave versions only.

dde advise *conversation_number* **item** name [**wait** *number_of_units* **second**[**s**]]

Notes

The `conversation_number` is the value returned by the `dde initiate` command.

The `wait` parameter overrides the delay specified by the `ddeTimeout` configuration setting. A maximum value of 65 seconds is possible.

This command must be used in conjunction with a `when DDE data` command, which defines a script to be executed each time the server passes a DDE DATA message with a new value for the item.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

dde execute

Sends a command string to be executed by the server. Available with Windows and NewWave versions only.

DDE execute *conversation_number* **command** command_string [**wait** *number_of_units* **second[s]**]

Notes

The conversation_number is the value returned by the dde initiate command.

The wait parameter overrides the delay specified by the ddeTimeout configuration setting. A maximum value of 65 seconds is possible. A timeout during the dde execute command does not necessarily mean that the server will not receive and process the action, only that the AdvanceLink script will not wait any longer for the acknowledgement proving that this has taken place.

Note also that DDE servers may respond to the dde execute command in different ways. Some respond as soon as they receive the request, while others respond only after they have completed execution.

See Chapter 5 for DDE examples.

Related Settings

ddeTimeout

dde initiate

Initiates a DDE conversation. Available with Windows and NewWave versions only.

dde initiate [**window** window_handle] [**application** name] [**topic** name] **into** *conversation_number*

Notes

If the window, application, and topic parameters do not exactly specify a single DDE respondent, the conversation is started with the first respondent only.

The integer value placed into the variable conversation_number is used to identify the conversation in all subsequent communications with the server.

See Chapter 5 for DDE examples.

Related Settings

ddeTimeout

dde poke

Sets the value of an item in the server application to the result of a specified expression. Available with Windows and NewWave versions only.

dde poke *conversation_number* **item** name **with** expression [**wait** *number_of_units* **second**[**s**]]

Notes

The `conversation_number` is the value returned by the `dde initiate` command.

The `wait` parameter overrides the delay specified by the `ddeTimeout` configuration setting. A maximum value of 65 seconds is possible. Note that a timeout during the `dde poke` command does not necessarily mean that the server will not receive and process the action, only that the AdvanceLink script will not wait any longer for the acknowledgement proving that this has taken place.

In some scripts it may be advantageous to issue DDE poke commands with a 0 length timeout to increase the speed of execution. Such a decision requires knowledge of how the server application processes the POKE message—some applications may deal with the initial requests slowly and thus reject later ones because the server is still busy.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

dde query

Obtains a list of currently-running applications that support DDE and/or the DDE topics they provide. Available with Windows and NewWave versions only.

```
dde query [application name] [topic name]
into variable
```

Notes

The response string is formatted with one application per line as follows:

```
window_handle, application_name, topic_name
```

If an `application` is specified, the command returns entries that correspond to that application. For example, if EXCEL is specified, the command returns a list of all instances of EXCEL which are running and support DDE.

If a `topic` is specified, the command only returns entries that correspond to that topic.

If neither `application` nor `topic` is specified, the command returns a list of all applications supporting DDE.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

dde request

Requests a data item from the server in the given conversation and places it into the given variable. Available with Windows and NewWave versions only.

dde request *conversation_number* **item** name **into** variable [**wait** *number_of_units* **second**[**s**]]

Notes

The `conversation_number` is the value returned by the `dde initiate` command.

The `wait` parameter overrides the delay specified by the `ddeTimeout` configuration setting. A maximum value of 65 seconds is possible.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

dde respond

Used in DDE handler scripts to send a message to the client application indicating the status of a DDE action it has submitted. In the case of DDE REQUESTs, used to return data. Available with WIndows and NewWave versions only.

dde respond accept | reject | busy | `expression`

Notes

Only one `dde respond` command should be issued by a handler script; if more than one is issued, only the first has effect. If a handler script terminates without issuing a `DDE respond` command, the "reject" response is sent to the client. Therefore, all DDE handler scripts should send a `dde respond`, except DDE TERMINATE handlers (you cannot respond "reject" or a "busy" to a TERMINATE request).

If a `dde respond` command is issued when the DDE engine is not expecting a response, the command is ignored.

`accept` means the handler is able to process the DDE action but has no data to return. This would be the normal response for all handlers except a DDE request handler which usually returns data. If `accept` is specified by a DDE REQUEST handler, an empty string is returned as data.

`reject` means the handler cannot process the DDE action at this time. This might be returned if the host application is shut down by an operator so that the data is no longer available, or if the request is deemed illegal by the script.

`busy` means the action requested is valid but cannot be processed at this time.

An `expression` can be returned only by a DDE REQUEST handler and is the data the handler retrieved. If this response is issued for any other type of handler, a runtime error occurs. While the expression does not have to yield a string, the data is always sent to the client in text form.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

dde terminate

Closes down the specified DDE conversation. Available with WIndows and NewWave versions only.

dde terminate *conversation_number*[**wait** *number_of_units* **second**[**s**]]

Notes

The `wait` parameter may be used to override the default delay specified by the `ddeTimeout` configuration setting. A maximum value of 65 seconds is possible. Note that a timeout during the DDE terminate command does not necessarily mean that the server will not receive and process the action, only that the AdvanceLink script will not wait any longer for the acknowledgement proving that this has taken place.

This command may be executed both when AdvanceLink acts as the client and when AdvanceLink acts as the server.

All DDE conversations are automatically terminated when the AdvanceLink window is closed and the program is shut down. The completion of a script does not shut down DDE links if the AdvanceLink program remains active.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

dde unadvise

Terminates monitoring of a server data item, established by the `dde advise` command. Available with WIndows and NewWave versions only.

`dde unadvise` *`conversation_number`* **`item`** `name` `[`**`wait`** *`number_of_units`* `time_unit]`

Notes

The `conversation_number` is the value returned by the `dde initiate` command.

The `wait` parameter overrides the delay specified by the `ddeTimeout` configuration setting. A maximum value of 65 seconds is possible.

See Chapter 5 for DDE examples.

Related Settings

`ddeTimeout`

delete file

Deletes an existing disk file.

delete file file_name

Notes

If the specified file does not exist, a runtime error occurs.

```
; Delete the file Testfile
delete file "Testfile"
```

Related Settings

directory

Errors

200 filePermissionError
201 fileNotFound
205 fileBusy

delete char

Deletes the character under the cursor and moves the rest of the line one space to the left.

delete char

delete line

Deletes the line the cursor is on and moves the rest of the lines on the screen up one line.

delete line

dial

Dials a phone number through an attached modem.

dial [phone_number_string] [**with** modem_commands]

Notes

If no phone_number_string is specified, the value of the phoneNumber configuration setting is used.

Any modem_commands specified are passed to the modem before the number is dialed, overriding AdvanceLink's default modem parameters. (See the Technical Information section of the *AdvanceLink Reference* manual for a list of the commands sent to the modem by default.)

```
; Send command to set modem's escape guard time with the phone number
dial "5124790735" with "ATS12=4"
```

Related Settings

baud
modemType
phoneNumber
phoneType
redial

Errors

260 modemNotResponding

disconnect

Terminates the connection to the current host.

disconnect

Related Settings

hardHangup

display

Displays the result of an expression on the terminal screen.

display expression

display file file_name

Notes

If an expression is specified, it may contain string or numeric data. The data is converted to a string before it is displayed.

file_name must be a PC-based file. When a file is specified, its contents are displayed on the terminal screen.

```
display "myfile.txt"      ; Displays the string myfile.txt
display myfile            ; Displays the contents of the variable myfile
display file "myfile.txt" ; Displays the contents of the file myfile.txt
display 3 + 2             ; Displays 5

; To display "Message" with inverse video display enhancement
display esc + "&dB" + "Message"
```

Errors

For display file only:

200 filePermissionError
201 fileNotFound
204 tooManyOpenFiles
202 fileIOError
205 fileBusy

Related Settings

alternateSet
autoHorizScroll
directory (For display file only)
display
displayFunctions
fontSize
formatMode

do

Performs a user-defined procedure, established with `proc` statement.

do `procedure_name`

```
proc saveSettings
  get baud into oldBaud
  get hostPrompt into oldPrompt
endproc

do saveSettings
```

do agent task

Runs a NewWave agent task.

do agent task [task_name] [**wait**]

Notes

If no `task_name` is specified, the HP NewWave object which has just been dropped or pasted into the window designates the task to be run.

TermTalk script execution always takes precedence over task execution. If the `wait` option is not specified, the agent task is unable to execute commands relating to the AdvanceLink window until the TermTalk script has completed. Agent task commands directed at other objects are carried out in parallel with the TermTalk script.

An error occurs if the specified task does not exist or an agent task is already running. The `agentTaskRunning()` function can be used to determine whether a task is running before issuing this command.

```
if not agentTaskRunning()
   do agent task "Backup"
endif
```

Errors

500 noCurrentObject
508 agentAlreadyRunning
509 agentTaskNotFound
510 currentObjectNotTask

do script

Invokes the specified script.

```
do script script_name

do script object object_name
```

Notes

Under Windows, `script_name` must be a fully-qualified script file name with either the `.TTS` or `.TTX` extension. If no path is specified, the file is looked for in the current directory. This is either the directory in which AdvanceLink is located, the directory selected in the last file-related dialog box, or the directory established with the `set directory` command. If the focused session is a NewWave session, the current directory will typically be `\HPNWPROG`.

The `do script object` form of the command operates on a NewWave object. `object_name` must be the name of a TermTalk object linked to the current session, that is, it must be listed in the Script Settings dialog box, displayed when you choose **Script** from the Settings menu.

After the invoked script terminates, control is passed to the statement following the `do script` command. `do script` commands can be nested to a depth of 15 levels.

```
; Execute script commands located in a file called TestScript:
ignore errors
do script "E:\W300\TESTSCRIPT.TTX"
if error()
   display "TestScript unsuccessful." + lf + cr
endif
```

Related Settings `directory`

Errors

380	notAScriptFile
200	filePermissionError
201	fileNotFound
202	fileIOError
204	tooManyOpenFiles
205	fileBusy
100	insufficientMemory

expect

Receives data from the host computer.

```
expect expected_string
[into variable] [for number_of_units
chunking_unit]] [timespec] [interactive]
```

where `chunking_unit` is one of the following:

```
character    characters
char         chars
line         lines
```

where `timespec` is one of the following:

```
wait number_of_units time_unit
wait until time_string
```

where `time_unit` is one of the following:

```
second       seconds
minute       minutes
hour         hours
```

where the format of `time_string` is:

```
"HH"
"HH:MM"
"HH:MM:SS"
```

Notes

`expected_string` specifies a string to be received from the host. (As with all string parameters, you may specify either a string literal enclosed in quotes or the name of a variable containing the string.) The command terminates upon receipt of this string.

The string received from the host can be stored into a variable by specifying a variable name after the `into` parameter. The text received may be limited to a specific number of characters or lines with the `for` parameter.

`timespec` specifies how long to wait for the expected string. It can be specified either as a number of seconds, minutes, or hours, or as a time of day (use 24-hour military clock). If no `timespec` is given, the number of seconds specified in the `timeout` configuration setting is used.

Normally the user cannot interact with the terminal while the `expect` command is waiting for a string. If the `interactive` keyword is specified, however, characters typed during the waiting period are sent to the host.

```
; Receive LISTF data from HP 3000 and display file names on screen:
sendline "listf @,2"
expect ":^Q" into listOfFiles# for 20 lines wait 10 seconds
i = 1
while i <= numberOfLines(listOfFiles#)
   currentFile# = word 1 of line i of listOfFiles#
   display currentFile# + lf + cr
   i = i + 1
endwhile
```

Related Settings `timeout`

Errors 270 expectTimeout

export object

Exports (or serializes) the current HP NewWave object to the named
MS-DOS file.

```
export object to file_name
```

Notes

If there is no current object, a runtime error occurs.

```
export object to "tempfile.ser"
```

Related Settings

`directory`

Errors

500 noCurrentObject
501 invalidDOSFilename
503 exportFailed

get

Retrieves information about the session from which the current script was run into a variable.


```
get [the] setting into variable

get [the] [screen] row into variable
get [the] [screen] column into variable

get [the] window_type window [location |
rectangle | title] into variable

get [the] fkey fkey_descriptor into variable

get [the] newWave_helpFile into variable
```

where `setting` is one of the following:

alternateSet
autoHorizscroll
autoKeyboardLock
autoLineFeed
backspaceKeyResult
baud
blockMode
blocksize
closeScript
compressXfer
connection
ctrlAltIsExtChar
cursorType
dataParityBits
dc1Pacing
ddeApplication
ddeServer
ddeTimeout
ddeTemplate
defaultBinaryXfer
directory
disableBell
disableHostControl
display
displayFunctions
draftPrintSize
emulation
enqAck
enterKeyResult
escDelayAmount
fontSize
formatMode
formsCache
graphResolution
graphScaling
graphWindows
hardHangup
helpFile
hostFileStartup
hostName
hostPrompt
inhibitDc2
inhibitHandshake
inhibitLineWrap
keyboardLock
language
leftMargin
lineModify
localEcho
lockFkeys
logDirection
maximizeSize
memoryLock
modemType
numberOfColumns
numberOfScreens
numericPlusKeyResult
objectScript
openScript
pageMode
phoneNumber
phoneType
port
recode8bitXfer
recodeCtrlXfer
redial
remote
returnKeyResult
rightMargin
sessionName
shiftBackspaceResult
smoothScroll
spacePerTab
tabKeyResult
terminalId
timeout
transferProtocol
transferTimeIncrement
typeAhead
waitHostPrompt
xmitFunctions
xonXoffInput
xonXoffOutput

where `window_type` is one of the following:
`text` `graphics` `script`

where `fkey` is one of the following:
`f1` `f2` `f3` `f4` `f5` `f6` `f7` `f8` `f9` `f10` `f11` `f12`

where `fkey_descriptor` is one of the following:
`label` `message` `script` `attribute`

Notes

The `setting` parameters allow you to retrieve the values of various configuration settings. For a complete description of each `setting`, see "Configuration Settings" later in this chapter.

The `row` and `column` parameters allow you to retrieve the memory-relative position of the cursor. To retrieve the screen-relative position, add the `screen` keyword to the command.

The `window` parameters allow you to get the current `location`, `rectangle`, and `title` of each window. `location` is returned as a string containing two comma-separated numbers. `rectangle` is returned as a string containing four comma-separated numbers.

The `fkey` parameters allow you to get the current values of the label, message, script, or attribute associated with a specific function key.

The `newWave_helpfile` parameter retrieves the name of the help file used by the AdvanceLink program.

```
; retrieve and store old baud rate prior to resetting it
get the baud into oldBaud#
set the baud to 1200

; do something like dial the modem, etc
set the baud to oldBaud#    ; restores the original baud rate
get the formatMode into oldFormatMode#

; This time we'll do something with the information
get the numberOfScreens into oldNumberOfScreens#
display "The current number of screens = " + oldNumberOfScreens# + lf + cr
get the fontSize into oldFontSize#
get the f1 label into f1Label#
```

Errors 104 getCommandFailed

graph

Performs the graphics functions listed below. Used only with versions of AdvanceLink which support graphics.

```
graph black
graph cursor [on|off]
graph cursor x_coordinate,y_coordinate
graph pen x_coordinate,y_coordinate
graph pen to cursor
graph draw
graph rbline [on|off]
```

Notes

`graph black` makes the entire graphics window black.

`graph cursor` with no parameters toggles the graphics cursor on and off.

`graph cursor` followed by two coordinates moves the graphics cursor to the specified location.

`graph pen` followed by two coordinates moves the graphics pen to the specified location. Coordinates specified are absolute pixel coordinates.

`graph pen to cursor` moves the graphics pen to the current cursor location. Specify absolute pixel coordinates.

`graph draw` draws a line from the pen to the cursor.

`graph rbline` with no parameters toggles the rubberband line on and off.

```
graph cursor on
graph cursor 10, 20      moves graphics cursor to given coordinate
graph pen 30, 65         ;moves the graphics pen to given coordinate
graph draw               ;draws a line from cursor to pen
```

Related Settings

`graphResolution`
`graphScaling`
`graphWindows`

hide

Hides the specified object from view.

```
hide controlBar
hide graphics
hide text
hide fkeys
hide indicators
hide script
```

Notes

A hidden object can be made visible again with the `show` command. If the object is already hidden, this command does nothing.

`hide controlBar` removes the control bar at the top of the window under the title bar; that area can then be used for text display.

`hide graphics` works only with versions of AdvanceLink which support graphics. In two-window mode, `hide graph` closes the graphics window. In one-window mode, it hides the graphics image.

`hide text` works only with versions of AdvanceLink which support graphics. In two-window mode, `hide text` closes the text window. In one-window mode, it hides the text portion of the display.

`hide fkeys` removes the on-screen function keys from the screen. The reclaimed lines are used to display terminal data.

`hide indicators` removes the row and column counters and other status indicators in the lower left corner of the window.

`hide script` closes the script window.

Related Settings

`lockFkeys` (For `hide fkeys` only)

ignore errors

Causes AdvanceLink to ignore non-fatal errors, so that no message is displayed and the script continues execution.

```
ignore errors
```

Notes

If a non-fatal error occurs, AdvanceLink's default is to report the error and terminate execution of all currently active scripts. However, you can override this default by specifying the `ignore errors` command. To return to default operation, specify `trap errors`. If scripts have been nested, `ignore errors` is in effect only for the current script.

```
Ignore errors                          ; Do not quit if an error occurs
open file "MYFILE"

if error()                             : If error occurs on open file
   create file "MYFILE"                ; file doesn't exist. Try to create it.
   if error()=203                      ; Check for the error "FileExists"
   display "That file already exists."
   stop
   endif
endif
trap errors                            ; Monitor errors again
```

import object

Imports a serialized HP NewWave object from MS-DOS.

```
import object [from] file_name [as
object_name]
```

Notes

If the command is successful, current object refers to the imported object. If the MS-DOS file does not exist or cannot be imported a runtime error occurs.

Normally, the name of the object is the same as that of the serial file. The `as` option can be used to specify a different object name.

```
import object from "tempfile.ser" as "Imported Object"
```

Errors

501 invalidDOSFilename
502 invalidObjectName
504 importFailed
505 renameFailed

input

Displays a dialog in which user can enter a text string, and saves response under a specified variable name.

input [**password**] [**into**] variable [**length** *length_of_input_string*] [**prompt** string] [**with** default_response]

Notes

The input command opens a dialog box in the center of the screen. The box contains a prompt, if specified, a box for text entry, and OK and Cancel buttons.

If OK is clicked or if the Return or Enter key is pressed, the contents of the text entry box is placed into the specified variable. If Cancel is clicked, an empty string ("") is placed into the variable.

password causes the text entered to be displayed as graphic characters for security purposes.

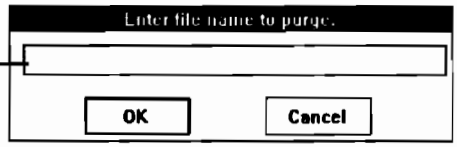
length specifies the maximum length of the input text. If this parameter is omitted, up to 255 characters are accepted.

Any default_response specified is placed in the text entry box and is highlighted.

The user's input is treated as string data. To use this data in a context that requires numeric data, the stringToNumber function must be used to convert the data type. See the last example for this command.

```
; Prompt for a file to be purged, specifying a maximum length of 8 characters
input into filename# length 8 prompt "Enter file name to purge:"
sendline "purge " + filename#
```

The preceding script displays the following box:

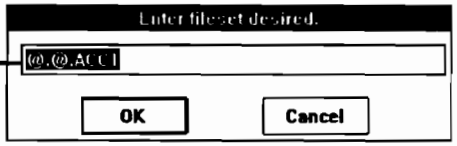


Only eight characters can be entered in this box.

```
; Send a user-specified fileset for use in obtaining a file listing on the host:
defaultFileSet# = "@.@.ACCT"
input into fileSet# prompt "Enter file set desired:" with defaultFileSet#
sendline "listf " + fileset#
```

The preceding script displays the following box:

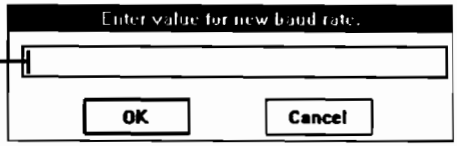
The default fileset is displayed here.




```
; Prompt user for a baud rate and convert to numeric type before setting baud
input into newBaud# prompt "Enter value for new baud rate:"
set the baud to stringToNumber(newBaud#)
```

The preceding script displays the following box:

The string entered here must be converted to a number.



insert char

Switches between insert character mode and overstrike character mode.

```
insert char [on] [with wrap]
insert char off
```

Notes

If `insert char` is specified with no parameters, it toggles between insert character mode and overstrike character mode.

The `with wrap` parameter can be specified when turning insert character mode on. This causes characters to wrap to the next line when insertion causes the line to reach the right margin.

```
insert char on              ; Turns insert mode on (it is off by default)
insert char off             ; Turns it off
insert char with wrap       ; Turns it on with wrap ("on" may be omitted)
insert char                 ; Toggles it off
insert char                 ; Toggles it on again
insert char on with wrap    ; Now wrap is on too
```

insert line

Inserts a line before the line the cursor is on and moves all lines below the new line down. The cursor is placed on the newly inserted line.

```
insert line
```

key

Equivalent to pressing the specified key the specified number of times.

key [**shift**] key_name [*number* [**times**]]

where key_name is one of the following:

backTab	**copyScreen**	**enter**	**PF1 - PF4**
f1 - f12	**lock**	**return**	**tab**
select	**stop**		

Notes

shift is used only with function keys f1 - f12 to transmit the value of a function key when it is used in combination with the Shift key.

PF1 - PF4 are recognized only when AdvanceLink is emulating a VT-100 terminal.

number must be a number from 1 through 255. If this parameter is omitted, the keypress is simulated once.

```
; Log on to a host HP 3000:
input into logonPhrase# prompt "Enter logon phrase:"
input password into myPassword# prompt "Enter password:"
key return 3 times ; get the attention of the host
sendline logonPhrase#
expect DC1
if numberOfChars(myPassword#) <> 0
  sendline myPassword#
endif
; Simulate pressing of the f9 key
key f9
```

Related Settings

emulation
enterKeyResult
keyboardLock
returnKeyResult
spacesPerTab
tabKeyResult

list files

Places a list of files into the supplied variable.

```
list files [with filespec] into variable
```

Notes

The list produced includes each file name in the current directory or folder, one per line.

`filespec` qualifies the file list with a file name and extension, such as `*.TXT` or `*.*`. MS-DOS wildcards `?` and `*` are both permitted.

```
; Obtain a list of the files in the current directory and display them:
list files into fileList#
if numberOfChars(fileList#) <> 0
   i = 1
   while i <= numberOfLines(fileList#)
      currentFile# = item 1 of line i of fileList#
      display currentFile# + lf + cr
      i = i + 1
   endwhile
i = i - 1
display lf + "There are " + i + " files in the current directory (" &
   + dirName() + ")" + lf + cr ; dirName() returns the directory name
endif
```

Related Settings

`directory`

log

Logs various parts of a user's session.

log all
log line
log page
log selection
start logging
stop logging
log pagebreak

Notes

Logged data can be printed with the `print log` command, saved to a disk file with the `save log` command, copied to the Clipboard with the `copy log` command, or cleared with the `clear log` command.

`log all` logs everything in display memory.

`log line` logs the entire line that the cursor is on.

`log page` logs the portion of display memory that is currently visible on the terminal.

`log selection` logs the current selection, if any. Use the `select` command to mark a selection from your script.

`start logging` activates automatic logging. Every line is logged until a `stop logging` command is executed. The log direction (top or bottom) is specified with the `logDirection` configuration setting .

The `stop logging` command turns off automatic logging.

The `log pagebreak` command inserts a formfeed character (ASCII 12) into the current log.

```
set logDirection top
start logging

; To stop logging later on
stop logging
; To save the log file
save log
```

Related Settings `logDirection`

maximize

Enlarges the specified window to occupy the entire monitor screen.

`maximize [text|graphics|script]`

Notes

The `graphics` parameter is supported only with graphics versions of AdvanceLink.

`text` is the default.

Related Settings

`maximizeSize`

message

Creates a message window with one to three buttons.

```
message [into variable] [prompt
prompt_string] [[icon] note|warn|stop]
buttons string [,string[,string]]
```

Notes

If a prompt string is supplied, it is displayed in the window. The string can be no longer than 4 lines, with each line containing up to 30 characters. You must supply carriage returns between lines by using the ^M control character (within a quoted string) or CR (used outside the quotes).

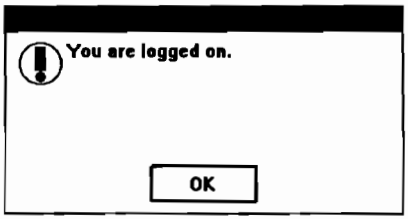
If a `variable` is specified, it receives a value representing the button pressed: 1 for the left most button, 2 for the center, or 3 for the rightmost. The rightmost button is highlighted for default selection with the Enter key.

If `note`, `warn` or `stop` is specified, the corresponding icon is displayed in the window: a question mark, exclamation point, or stop sign respectively.

Button strings can be no longer than 9 characters. At least one button must be specified.

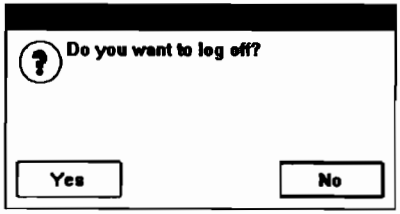
```
; Display a simple message
beep
message prompt "You are logged on." icon warn buttons "OK"
```

The preceding script displays the following box:



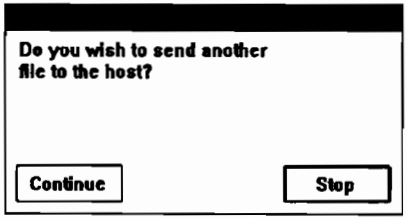
```
; Test user's response to a message
beep
message into response# prompt "Do you want to log off?" icon note &
buttons "Yes","No"
if response = 1
   disconnect
endif
```

The preceding script displays the following box:




```
; Display a long message, including carriage returns
message into response# prompt "Do you wish to send another" + CR &
   + "file to the host?" buttons "Continue","Stop"
; Or you can do it this way
message into response# prompt "Do you wish to send another^Mfile to the &
    host?" buttons "Continue","Stop"
```

Either of the message commands in the preceding script would display the following box:



minimize

Collapses the AdvanceLink application to an icon. Windows only.

```
minimize
```

open application

Opens an application, or opens a file using the specified application.

```
open application [maximized|minimized]

open application with file [maximized | minimized]
```

Notes

In a multi-application environment, the application is opened concurrent with AdvanceLink. In a single-tasking environment, AdvanceLink terminates and the application is opened. When the application quits, AdvanceLink is restarted but the script does not continue execution.

The `maximized` and `minimized` keywords are only supported on the Windows platform. If the `maximized` keyword is specified, the window of the opened application covers the entire screen. If the `minimized` keyword is specified, the application is launched as an icon. If neither `maximized` or `minimized` is specified, the application is opened into a standard-sized window.

```
open "PageMaker"
open "Microsoft Word 4.0" with "MyReport"
```

Errors

390	unableToOpenApplication
201	fileNotFound
205	fileBusy
100	insufficientMemory

open file

Opens an ASCII text file on the PC for reading and writing.

```
open file file_name
```

Notes

A fully-qualified file name can be specified, with or without a file extension. If not otherwise specified, AdvanceLink looks for the file in its own directory, unless this default has been overriden by setting the `directory` configuration parameter or by changing the current directory in any Windows file selection dialog box.

Two file pointers are maintained for the file: one for reading and one for writing. Initially, the read file pointer is positioned at the beginning of the file. Writing always takes place at the end of the file.

```
; Open a user-specified file for read/write:
input into filename# prompt "Enter desired file name:"
if filename# = ""
   stop
endif
open file filename#
```

Errors

200	filePermissionError
201	fileNotFound
204	tooManyOpenFiles
205	fileBusy
209	fileAlreadyOpen

Related Settings

`directory`

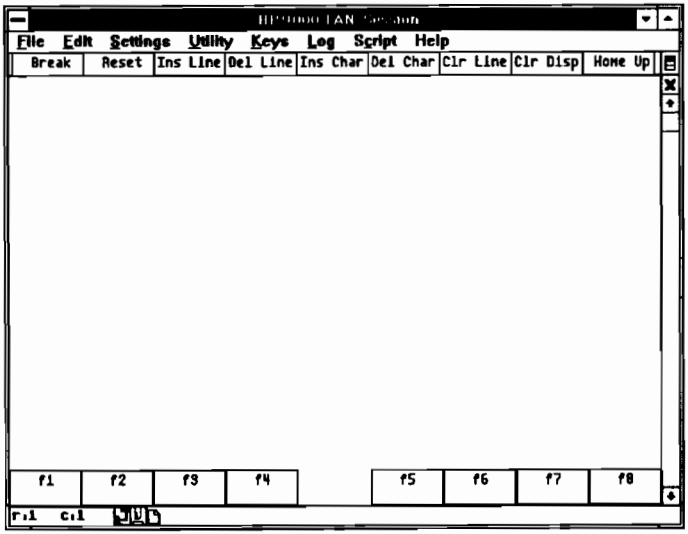
open session

Opens a session file, retrieves settings and, on the Macintosh platform, reads any text and graphics saved in the file into display memory.

open session file_name

```
open session "HP9000 LAN Session"
```

The preceding command would open a window like this:



Related Settings `openScript`

Errors

200	filePermissionError
201	fileNotFound
202	fileIOError
204	tooManyOpenFiles
205	fileBusy
100	insufficientMemory
361	badConfigFile

page setup

Displays a dialog box in which the user can specify page setup options. Macintosh only.

```
page setup
```

paste

Pastes the contents of the clipboard into display memory at the current cursor location, or into the specified variable.

```
paste [into|after variable]
```

Notes

To paste at the current cursor location, specify `paste` with no additional parameters.

When pasting to a variable, specifying `into` replaces any data currently contained in that variable. Specifying `after` appends the data to the end of the current contents of the variable.

```
; Copy line 1 and paste its contents to line 10
select line 1
copy
cursor 10
paste
; Copy line 1 and paste it into the variable mylines#
select line 1
copy
paste into mylines#
; Select another line and append it to the variable
select line 10
copy
paste after mylines#
; Now mylines# contains lines 1 and 10
```

Errors

391 errorAccessingClipboard

print

Prints a text file, the graphics screen, or the current log to the printer currently specified in the Windows Control Panel or Macintosh Chooser..

```
print file file_name
print graph
print log
```

```
:To print the contents of the file "myfile.txt"
print file "myfile.txt"
; Display a dialog box from which the user can print a file, the contents of the
; graphics window, or the current log.
message into printChoice# prompt "What would you like to print" &
   buttons "FILE", "LOG", & "GRAPH"
if printChoice#=1
; Use getfilename function to display a standard file open box.
; *.* means all files will be displayed.

   print file getfilename("File to print:","*.*")
elseif printChoice#=2
   print log
else
   print graph
endif
```

Related Settings

`directory` (for `print file` only)
`draftPrintSize` (for `print log` only)

Errors

350	printError
200	filePermissionError
201	fileNotFound
202	fileIOError
204	tooManyOpenFiles
205	fileBusy
100	insufficientMemory

quit

Terminates execution of AdvanceLink, as if **Quit** or **Exit** were selected from the File menu.

```
close
exit
quit
```

Related Settings `closeScript`

read file

Reads from the current read file pointer of a disk file previously opened with the `open file` command or the `create file` command.

read file file_name [read_condition] **into** variable

where read_condition is one of the following:
until string
for *number_of_units* chunking_unit

where chunking_unit is one of the following:

characters	**character**
char	**chars**
line	**lines**

Notes

The file name specified must exactly match the file name that was specified in the `open file` command or the `create file` command.

Reading continues until the read_condition is met. If no condition is specified, one line is read by default.

The string following the `until` keyword specifies a string of terminating characters. Reading stops when these characters are read. If the string is the empty string (""), the command reads until the end of the file.

The expression following the `for` keyword specifies how many characters or lines to read.

Reading from a file where the read pointer is positioned at the end of file returns an empty string.

```
; Read a user-specified file into display memory
fileName# = getFileName("File to display:", "*.*")
if fileName# = ""
   stop
endif
open file fileName#
read file fileName# for 1 line into currentLine#
while currentLine# <> ""
   display currentLine# + lf + cr
   read file fileName# for 1 line into currentLine#
endwhile
close file fileName#

; Read the data file one field at a time. The fields are comma separated.
; Display the fields one per line.
open file "MYDATA"
read file "MYDATA" until "," into currentField#
while currentField# <> ""
   display currentField# + cr + lf
   read file "MYDATA" until "," into currentField#
endwhile
close file "MYDATA"
```

Related Settings `directory`

Errors

200	filePermissionError
201	fileNotFound
202	fileIOError
205	fileBusy
208	fileNotOpen

receive

Transfers a file from the host (the remote file) to the workstation (the local file).

```
receive [protocol link3000 | link9000 |
xmodem]
```

For transfer from an HP 3000:

```
receive remote_file [to local file] [as text
| wordwrap | binary | backup | restore |
newwavepackage] [delete] [protocol link3000]
```

For transfer from an HP 9000:

```
receive remote_file [to local file] [as text
| binary | backup | restore | newwavepackage]
[delete] [protocol link9000]
```

For xmodem transfer :

```
receive remote_file [to local file] [delete]
[protocol xmodem]
```

Notes

If no parameters are specified, `receive` operates as if it had been selected from the File menu, displaying a dialog box in which the user can enter a file name and specify conversion options. The transfer is performed using the protocol specified by the `transferProtocol` configuration setting, which defaults to `link3000`.

If only the `protocol` parameter is specified, the command operates interactively using the specified protocol.

If the local file name is omitted, the local file takes the same name as the remote file. If a local file with the given name already exists, a runtime error occurs unless `delete` is specified to overwrite the existing file.

File conversion optionsare available for transfers from the HP 3000 or HP 9000. If the `as` parameter is omitted, the default conversion option depends on the type of file transferred. The following chart shows the options available for various file types; the asterisked option is the default. "PC Backup" and "NewWave Backup" files are created using the backup option of the `send` command.

Option	Text File	PC Backup	NewWave Backup	Other File
text	Yes*	No	No	No
binary	Yes	Yes	Yes	Yes*
wordwrap	Yes	No	No	No
backup	Yes	Yes	No	Yes
restore	No	Yes*	No	No
newwave	No	No	Yes*	No

When used for HP 3000 transfers, the `text` option converts an HP 3000 EDIT/3000 file into standard PC text format. All trailing blanks are stripped from each record, and a single carriage return is appended. Non-printing characters, except tab, are replaced by a space. The default file extension `.TXT` is automatically added to the file name.

When used for HP 9000 transfers, the `text` option converts a text file, produced by many HP-UX editors, into a PC standard text file. In particular, it changes a newline, linefeed, or <LF> character, used to separate lines and paragraphs by HP-UX editors, into carriage-return <CR> and linefeed <LF> characters, used to separate lines and paragraphs by PC applications. The default file extension `.TXT` is automatically added to the file name.

The `wordwrap` option, available for HP 3000 transfers only, converts an HP 3000 EDIT/3000 file into standard PC text format. To perform word wrap, all leading and trailing blanks are stripped from each record in the HP 3000 file and a single blank is appended. When a blank record is encountered in the HP 3000 text file, a single carriage return is placed in the converted PC file to mark the end of a paragraph. Non-printing characters are replaced by a space. The default file extension `.TXT` is automatically added to the file name.

The `binary` option copies any type of HP 3000 or HP 9000 file into a PC file. All data characters in each record of the HP 3000 file are written to the data file with no modifications. The default file extension `.TXT` is automatically added to the file name.

The `backup` option copies any type of HP 3000 or HP 9000 file onto a PC so that it can later be restored with the `restore` option of the `send` command. All data and user labels are copied, so that all file characteristics are preserved when the file is restored to the HP system. The file extension `.HP3` or `.HP9` is automatically added to the file name.

The `restore` option restores a PC file which has previously been archived on the HP 3000 or HP 9000 with the `backup` option of the `send` command. Data and file attributes are all restored.

The `newwave` option restores a NewWave object which has previously been archived on the HP 3000 or HP 9000 with the `newwave` option of the send command.

Additional details on file transfer are found in the *AdvanceLink Reference* manual.

```
receive "myfile"                        ; receives myfile using default protocol
                                        ; placing it on PC with same name

receive "myfile" to "newfile" delete    ; receives myfile using default protocol
                                        ; placing it on PC as newfile, deleting an
                                        ; old copy of newfile if it already exists

receive "myfile" as binary              ; receives myfile using default protocol
                                        ; using the binary conversion option

receive "myfile" protocol link9000      ; receives myfile  using 9000 protocol
```

Related Settings

```
baud
blockSize
compressXfer
dc1Pacing
defaultBinaryXfer
hostFileStartup
recode8BitXfer
recodeCtrlXfer
transferProtocol
transferTimeIncrement
```

Errors

100	insufficientMemory
105	stringTooLong
200	filePermissionError
202	fileIOError
203	fileExists
204	tooManyOpenFiles
205	fileBusy
206	fileTooLarge
207	diskFull
300	unableToRunLinkProgram
303	fileTransferUnsuccessful
304	fileTransferCancelled
305	hostfilePermissionError
306	fileNameTooLong
308	hostfileNotFound
307	invalidLinkVersion

receive object

Receives a file from the host and imports it as an HP NewWave object.

receive object [**from**] remote_file [**protocol** **link3000** | **link9000** | **xmodem**]

Notes

If the command is successful, current object refers to the imported object.

The title of the object once it has been imported into the NewWave Office will be that contained in the serialized object file.

If `protocol` is not specified, the protocol specified by the `transferProtocol` configuration setting is used. This defaults to `link3000`.

```
; Receives myfile using default protocol
receive object from "myfile"
```

Related Settings

`baud`
`blockSize`
`compressXfer`
`dclPacing`
`defaultBinaryXfer`
`hostFileStartup`
`recode8BitXfer`
`recodeCtrlXfer`
`transferProtocol`
`transferTimeIncrement`

Errors

504 importFailed
506 hostFileNotObject

rename file

Renames a file on disk.

```
rename file old_file_name to new_file_name
```

Notes

`old_file_name` must evaluate to the name of an existing disk file.

`new_file_name` must evaluate to a file name valid for the operating system. If a file with this name exists, an error occurs.

```
; Rename a user-specified file:
oldName# = getFileName("File to rename:", "*.*")
if oldName# = "" or not exist(oldName#)
   stop
else
   newName# = newFileName( "New file name:", "")
   if newName# = ""
      stop
   else
      rename file oldName# as newName#
   endif
endif
```

Related Settings

`directory`

Errors

200 filePermissionError
201 fileNotFound
203 fileExists
205 fileBusy

reset

Resets the terminal.

```
soft reset
hard reset
```

Notes

`soft reset` emulates the function of the Reset key on an HP terminal. AdvanceLink's soft reset does the following:

- Display Functions and Keyboard Lock are turned off.
- Any data communication transfers in progress are cancelled.
- The bell rings.
- The screen is refreshed.

`hard reset` has the same function as pressing Ctrl + Shift + Reset on an HP terminal. AdvanceLink's hard reset does the following:

- Display memory is cleared.
- All tabs are cleared. (Margins are not cleared.)
- The following configuration values and mode settings are reset to their default values:
  - Page mode off
  - Inhibit Line Wrap (strap C) off
  - Inhibit Handshake (strap G) off
  - Inhibit DC2 (strap H) off
  - Transmit Functions off
  - Keyboard Lock off
  - Memory Lock off
  - Insert Character mode off
  - Display Functions off
- The typeahead buffer is cleared.
- Any datacomm transfers in progress are canceled and the buffers are emptied.
- Display Functions and Keyboard Lock are turned off.
- If enabled, record mode and logging are disabled.
- The bell rings.
- The screen is refreshed.

Related Settings

`emulation`

restore

Restores AdvanceLink's windows to their normal size. Used under Microsoft Windows only.

`restore [text|graphics|script]`

Notes

If the AdvanceLink application is iconized, the application is restored — all open windows are visible and the window specified in the `restore` command is brought to the front. The default is `text`.

The `graphics` parameter is supported only with graphics versions of AdvanceLink.

return

Terminates execution of current procedure or current script. If the current script was initiated by another script, returns control to initiating script.

`return`

```
proc printProcedure
   fileName# = getFileName("File:", "*.*")
   if fileName# = "" or not exist(fileName#)
      return
   endif
   print file fileName#
endproc

do printProcedure

; when return in the above procedure is encountered, or when the procedure
; completes, script execution resumes at the following line
display "Complete"
```

revert

Reads the current session file, returns all settings to the values specified therein, clears display memory, and performs a hard reset.

```
revert
```

Notes If no session file is open, AdvanceLink's default configuration settings are restored.

Errors

100	insufficientMemory
200	filePermissionError
201	fileNotFound
202	fileIOError
204	tooManyOpenFiles
205	fileBusy
361	badConfigFile

save

Saves AdvanceLink data and/or settings to a file.

```
save text file_name [delete]
save graph file_name [as paint|draw] [delete]
save log file_name [delete]
save session file_name [delete ]
```

Notes

`save text` saves the current contents of display memory to a disk file. If you add the extension `.TXT` to the file name you specify, the file can be opened by most Windows text processors.

`save graphics` saves the current contents of graphics memory to a disk file. If `paint` or `draw` is specified, the graphics are saved as a painting or drawing file respectively. Default is `paint`.

`save log` saves the current log to a disk file.

`save session` saves the current configuration settings (all `set` command parameters and all options chosen using commands from AdvanceLink's Settings menu) to a disk file.

In all forms of the command, the `delete` keyword can be specified to overwrite an existing file with the name specified. With save log, if the file exists and delete is not specified, the newly logged data is appended to the end of the file. With the other versions of the command, if the file exists and `delete` is not specified, a runtime error occurs.

```
; Save the contents of display memory to a file:
saveFileName# = newFileName("New File:","")
if saveFileName# <> ""
   save text saveFileName#
endif

; Change some settings and save the session to a file:
set baud to 1200
set numberofColumns to 132
save session " MySessionName" delete
```

Related Settings `directory`

Errors

100	insufficientMemory
200	filePermissionError
201	fileNotFound
202	fileIOError
203	fileExists
204	tooManyOpenFiles
205	fileBusy
206	fileTooLarge
207	diskFull

select

Selects or deselects parts of display memory.

select select_option

where select_option is one of the following:
upper_row, left_col, lower_row, right_col
line [*number* [**to** *number*]]
screen
all
cancel

Notes

A rectangular region of the screen can be specified by specifying four screen coordinates. The coordinates must be given in the order shown above. For example, `select 1,1,23,80` would select the entire screen except the last line.

A single line or a range of lines can be specified with the `line` option. If `line` is specified alone, the line where the cursor is currently positioned is selected.

`select screen` selects the visible portion of display memory.

`select all` selects the entire contents of display memory.

`select cancel` deselects any current selection.

```
; Select some lines in display memory and copy them to the clipboard
select line 1 to 10
copy
```

send

Transfers a file from the personal computer (the local file) to the host (the remote file).

```
send [protocol link3000 | link9000 | xmodem]
```

For transfer to an HP 3000:

```
send local_file [to remote_file] [as text |
wordwrap | binary | backup | restore |
newwavepackage] [delete] [protocol link3000]
[record number bytes|words]
```

For transfer to an HP 9000:

```
send local_file [to remote_file] [as text |
binary | backup | restore | newwavepackage]
[delete] [protocol link9000]
```

For xmodem transfer :

```
send local_file [to remote_file] [delete]
[protocol xmodem]
```

Notes

If no parameters are specified, `send` operates as if it had been selected from the File menu, displaying a dialog box in which the user can enter a file name and select conversion options. The protocol used is that specified by the `transferProtocol` configuration setting, which defaults to `link3000`.

If only the `protocol` parameter is specified, the command operates interactively using the specified protocol.

If `remote_file` is omitted, the remote file name is built from the local file name, using the first eight characters that are valid for a file name on the host operating system. If a remote file with the given name already exists a runtime error occurs, unless `delete` is specified to overwrite the existing file.

For HP 3000 transfers only, `record` specifies the size for the remote file in either bytes or words.

File conversion options are available for transfers to the HP 3000 or HP 9000. If the `as` parameter is omitted, the default

conversion option depends on the type of PC file transferred. The following chart shows the options available for various file types; the asterisked option is the default. "HP Backup" files are created using the `backup` option of the `receive` command.

Option	PC Text File	HP Backup	NewWave Object	Other File
text	Yes*	No	No	No
binary	Yes	Yes	Yes	Yes
wordwrap	Yes	No	No	No
backup	Yes	No	No	Yes*
restore	No	Yes*	No	No
newwave	No	No	Yes*	No

When used for HP 3000 transfers, the `text` option converts a PC text file into standard HP EDIT/3000 format. By default, the records are 72 bytes, ASCII. Each line in the PC file (terminated by a carriage return) is written to a single record in the HP 3000 file. If a line is too long to fit in the established record length, as many characters are placed in the record as will fit. Additional records are written until all characters have been copied. Carriage returns are stripped from the PC file. When consecutive carriage returns are encountered, each one after the first causes a blank record to be written to the EDIT/3000 file. All other non-printing characters including new-line, new-page, and so on, are replaced by a space. Tabs, however, are not replaced with a space.

When used for HP 9000 transfers, the `text` option converts a PC text file into a file suitable for use with HP-UX editors such as `vi`. In particular, it converts a text file which uses the carriage return <CR> and linefeed <LF> characters to separate lines or paragraphs into a file which uses the linefeed <LF> character to separate lines or paragraphs. This is the default option for text files.

The `wordwrap` option is available for transfers to the HP 3000 only. This option converts a PC text file into standard HP EDIT/3000 format. By default, the records are 72 bytes, ASCII. To perform wordwrap, each record of the resulting text file is filled with as many whole words (text separated by spaces or non-printing characters) as will fit in the established record length. Carriage returns are stripped from the PC file. When consecutive carriage returns are encountered, each one after the first causes a blank record to be written to the file. All other non-printing characters, including tab, new-line (LF), new-page (FF), and so on, are replaced by a space.

The `binary` option copies any type of PC file into an HP 3000 or HP 9000 binary file. Only the data is copied; the file attributes are not. The data in the file is sent to the host with no modifications. On the HP 3000, the default record length is 128 words, binary, and each record is completely filled with data. For example, if the HP 3000 record length is 128 words, the first 256 characters (bytes) in the PC file are written to the first record; the next 256 characters are written to the second record, and so on.

The `backup` option copies any type of PC file onto the HP 3000 or 9000 so that it can later be restored with the `restore` option of the `receive` command. Data and file attributes are copied, so that all characteristics are preserved when the file is restored.

The `restore` option restores an HP 3000 or 9000 file previously archived on the PC with the `backup` option of the `receive` command.

The `newwave` option translates a NewWave object into Serialized Object Format and archives it on the HP 3000 or 9000, so that it can later be restored with the `receive` command.

Additional details on file transfer are found in the *AdvanceLink Reference* manual.

```
send "myfile"                          ; sends pc-based myfile to host
                                       ; using default protocol, and
                                       ; places it in a file with the same name

send "myfile" to "newfile" delete      ; sends myfile using default protocol
                                       ; placing it on host as newfile, deleting
                                       ; old copy of newfile if it already exists

send "myfile" as binary                ; sends myfile using default protocol
                                       ; using the binary conversion option

send "myfile" protocol link9000        ; sends myfile using link9000 protocol

send "myfile" record 80 bytes          ; sends myfile and places it on host in a
                                       ; file called "myfile" with 80 byte records
```

Related Settings

`baud`
`blockSize`
`compressXfer`
`dc1Pacing`
`defaultBinaryXfer`
`hostFileStartup`
`recode8BitXfer`
`recodeCtrlXfer`
`transferProtocol`
`transferTimeIncrement`

Errors

100	insufficientMemory
105	stringTooLong
200	filePermissionError
201	fileNotFound
202	fileIOError
203	fileExists
204	tooManyOpenFiles
205	fileBusy
300	unableToRunLinkProgram
301	illegalConversionOption
302	illegalRecordSpecification
303	fileTransferUnsuccessful
304	fileTransferCancelled
305	filePermissionError
306	fileNameTooLong

309 hostFileExists
310 invalidRecordSize
307 invalidLinkVersion

sendline

Sends a string to the host followed by a carriage return.

sendline string

Notes

To receive a response from the host, use the expect command immediately following sendline.

If multiple lines of data are sent, and if the waitHostPrompt configuration parameter is on, the second and subsequent sendline or sendstring commands in a script wait for a host prompt before sending data. If waitHostPrompt is off, neither command waits for the host prompt. The host prompt is set with the hostPrompt configuration parameter.

The maximum length of the string sent is 32,767 characters.

```
; Sending data to the host:
input into logonString# prompt "Enter your logon command:"
input password into passwordString# prompt "Enter your password:"
set hostPrompt to dc1
sendline logonString#
expect dc1 wait 10 seconds
if passwordString# <> ""
   sendline passwordString#
   expect dc1
endif
input into command# prompt "Enter command to be performed:"
if command# <> ""
   sendline command#
endif
sendline "bye"
```

Related Settings

autoLineFeed
escDelayAmount
hostprompt
localEcho
typeAhead
waitHostPrompt

Errors

105 stringTooLong

sendstring

Sends a string to the host without a carriage return.

sendstring string

Notes

To receive a response from the host, use the `expect` command immediately following `sendstring`.

If multiple lines of data are sent, and if the `waitHostPrompt` configuration parameter is on, the second and subsequent `sendline` or `sendstring` commands in a script will wait for a prompt from the host before sending data. If `waitHostPrompt` is off, neither command waits for the host prompt. The host prompt is set with the `hostPrompt` configuration parameter.

The maximum length of the string sent is 32,767 characters.

```
; This command can be used to build a command from separate parts
sendstring "LISTF "
sendstring ",2"
sendline ""
```

Related Settings

`autoLineFeed`
`escDelayAmount`
`hostprompt`
`localEcho`
`typeAhead`
`waitHostPrompt`

Errors

105 stringTooLong

send object

Transmits the current HP NewWave object to the host computer.

```
send object [to remote_file_name] [delete]
[protocol link3000 | link9000 | xmodem]
```

Notes

If no `remote_file_name` is specifed, a default name is created from the object name, using as many letters as are permissible for a file name under the host operating system.

The `delete` option can be used to overwrite an existing remote file.

If the `protocol` option is not specified, the protocol specified by the `transferProtocol` configuration setting is used. This defaults to `link3000`.

If there is no current object , a runtime error occurs.

```
; Sends the object just pasted or dropped as NewWave SOF package
send object to "myfile"
```

Related Settings

```
baud
blockSize
compressXfer
dc1Pacing
defaultBinaryXfer
hostFileStartup
recode8BitXfer
recodeCtrlXfer
transferProtocol
transferTimeIncrement
```

Errors

500 noCurrentObject
503 exportFailed
507 pcFileNotObject

set

Changes a setting to the value specified. Applies to the focused session (the session from which the current script was run) only.

set [**the**] setting **to** expression

Settings are listed on the next page.

set [**the**] [**screen**] **row to** expression

set [**the**] [**screen**] **column to** expression

set [**the**] window_type **window location to** location_string

set [**the**] window_type **window rectangle to** rectangle_string

set [**the**] window_type **window title to** title_string

where window_type is one of the following:
text **graphics** **script**

where location_string provides the coordinates of the upper left corner of the window in this format:
"upper_ pixel,left_pixel"

where rectangle_string provides the coordinates of the upper left and lower right corners of the window in this format:
"upper_pixel,left_pixel,lower_pixel,right_pixel"

set [**the**] fkey fkey_descriptor **to** expression

where fkey is one of the following:
f1 **f2** **f3** **f4** **f5** **f6** **f7** **f8** **f9** **f10** **f11** **f12**

where fkey_descriptor is one of the following:
label **message** **script** **attribute**

set **[the]** newWave_helpFile to expression

where `setting` is one of the following:

alternateSet	enterKeyResult	numericPlusKeyResult
autoHorizscroll	escDelayAmount	objectScript
autoKeyboardLock	fontSize	openScript
autoLineFeed	formatMode	pageMode
backspaceKeyResult	formsCache	phoneNumber
baud	graphResolution	phoneType
blockMode	graphScaling	port
blocksize	graphWindows	recode8bitXfer
closeScript	hardHangup	recodeCtrlXfer
compressXfer	helpFile	redial
connection	hostFileStartup	remote
ctrlAltIsExtChar	hostName	returnKeyResult
cursorType	hostPrompt	rightMargin
dataParityBits	inhibitDc2	sessionName
dc1Pacing	inhibitHandshake	shiftBackspaceResult
ddeApplication	inhibitLineWrap	smoothScroll
ddeServer	keyboardLock	spacePerTab
ddeTimeout	language	tabKeyResult
ddeTemplate	leftMargin	terminalId
defaultBinaryXfer	lineModify	timeout
directory	localEcho	transferProtocol
disableBell	lockFkeys	transferTimeIncrement
disableHostControl	logDirection	typeAhead
display	maximizeSize	waitHostPrompt
displayFunctions	memoryLock	xmitFunctions
draftPrintSize	modemType	xonXoffInput
emulation	numberOfColumns	xonXoffOutput
enqAck	numberOfScreens	

Notes

The `setting` parameters allow you to set various configuration items. For a complete description of each `setting`, see "Configuration Settings" later in this chapter.

The `window` parameters allow you to change the current `location`, `rectangle`, and `title` of each window.

The `row` and `column` parameters allow you to set the memory-relative position of the cursor. To set the position of the cursor relative to the portion of memory currently displayed on the `screen`, add the screen keyword.

`set fkey` allows you to change the current values of the label, message, script, or attribute associated with a specific

function key. Valid attributes include `normalAttribute`, `localAttribute`, `transmitAttribute`, or `scriptAttribute`.

The `newWave_helpFile` parameter allows you to change the HP NewWave help file used by AdvanceLink.

```
; Save current config setting, and then set new one
proc saveSettings
   get the baud into oldBaud#
endproc

proc changeSettings
   set the baud to stringToNumber(newBaud#)
   set the numberOfScreens to stringToNumber(newNumberOfScreens#)
endproc

do saveSettings
input into newBaud# prompt "Enter value for new baud rate:" with &
   numberToString(oldBaud#)
input into newNumberOfScreens# prompt "Enter new number of screens:" &
   with numberToString(oldNumberOfScreens)
do changeSettings

; Using set to size text window
input into response# prompt &
   "Enter new rectangle coordinates: four numbers, separated by commas:"
upperRow# = stringToNumber(item 1 of response#)
leftCol# = stringToNumber(item 2 of response#)
lowerRow# = stringToNumber(item 3 of response#)
rightCol# = stringToNumber(item 4 of response#)
set the text rectangle to upperRow#,leftCol#,lowerRow#,rightCol#
input into newTitle# prompt "Enter new graph window title:"
if numberOfChars(newTitle#) <> 0
   set the graph title to newTitle#
endif

; Using set to attach a script to a function key, and provide appropriate label
set f1 script to "logon.TTS"
set f1 label to "LOGON"
```

Errors 102 invalidSetValue
103 setCommandFailed
105 stringTooLong

setup

Displays a dialog box in which the user can change AdvanceLink's configuration.

setup function [**result** variable]

where function is one of the following:

color
connection
enhancements (Macintosh only)
fkeys
graphics
keyboard
logging
scripts
terminal
transfer

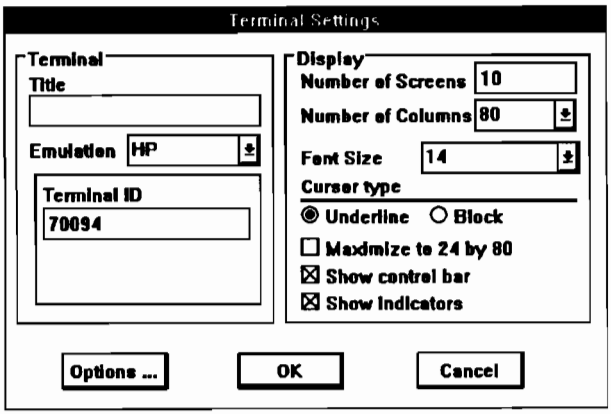
Notes

To monitor whether the user selected the **OK** or **Cancel** button on the dialog box, specify the result keyword. If **OK** is selected, a value of 1 is placed in the variable. Otherwise, it contains a value of 0.

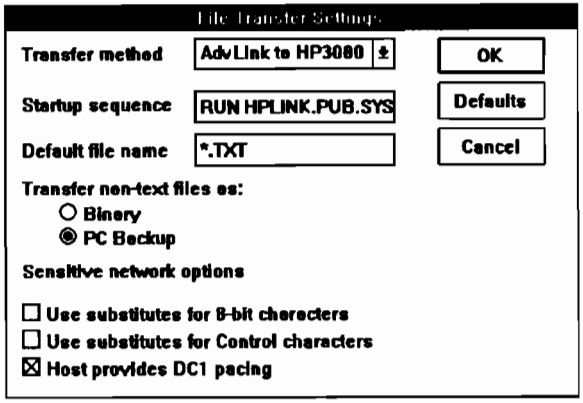
```
; Display dialogs to allow user to change configuration settings.
; If any changes are made, save settings in a session file.
setup terminal result response1
setup transfer result response2
setup keyboard result response4
if response1=1 or response2=1 or response3=1 or response4=1
   save session "newconfig"
endif
```

The preceding script displays the dialog boxes shown on the following pages, recording any configuration changes made to a session file. See the *AdvanceLink Reference* manual for complete details on these dialog boxes.

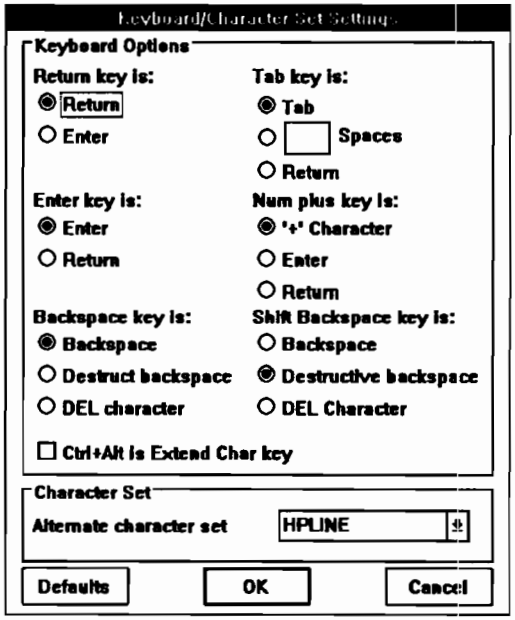
Terminal Settings:



File Transfer Settings:



Keyboard Settings:



show

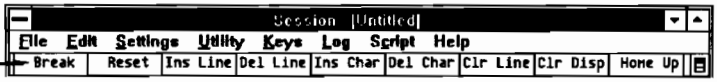
Makes the specified display element visible to the user.

```
show controlBar
show graphics
show text
show fkeys
show indicators
show script
```

Notes

Objects are hidden with the `hide` command. If the specified object is already visible, this command does nothing.

`show controlBar` displays the buttons under the menu bar.



`show graphics` works only with versions of AdvanceLink which support graphics. In two-window mode, it opens the graphics window, or brings it to the front. In one-window mode, it displays the graphics image.

`show text` works only with versions of AdvanceLink which support graphics. In two-window mode, `show text` opens the text window, or brings it to the front. In one-window mode, it displays the text.

`show fkeys` displays the on-screen function keys.

`show indicators` displays the row and column counters, connection, keyboard lock, logging and script indicators at the lower left of the window.

`show script` opens the script window.

Related Settings

`lockFkeys` (For `show fkeys` only.)

stop

Terminates execution of all currently-active scripts

```
stop
```

```
; Stop execution if the user does not input a file name in response to prompt
input into filename# prompt "Enter desired file name:"
if filename# = ""
   stop
endif
open file filename#
```

tab

Sets a tab at the specified column.

```
tab [at column_number]
```

Notes

If no column is given, a tab is set at the current cursor column.

```
; Set a special group of tab settings:
MyTabSettings# = "7,8,12,16,20,24,28,32,36"
i = 1
while i <= numberOfItems(MyTabSettings#)
   thisTab# = stringToNumber(item i of MyTabSettings#)
   tab at thisTab#
   i = i + 1
endwhile
```

trap errors

Reinstates error trapping after `ignore errors` has been specified, so that whenever an error is found, a message is displayed and the script terminates.

```
trap errors
```

Notes

If a non-fatal error occurs, AdvanceLink's default is to report the error and terminate execution of all currently active scripts. However, you can override this default by specifying the `ignore errors` command. To return to default operation, specify `trap errors`. If scripts have been nested, `ignore errors` will remain in effect until the highest level script terminates, or until a `trap errors` statement is encountered.

```
ignore errors                          ; Do not quit if an error occurs
open file "MYFILE"

if error()                             ; If error occurs on open file
   create file "MYFILE"                ; tile doesn'texist. Try to create it.
   if error()=203                      ; Check for the error "FileExists"
   display "That file already exists."
   stop
   endif
endif
trap errors                            ; Monitor errors again
```

wait

Pauses script execution until a specified time.

wait until time_string

where the format of time_string is
"HH"
"HH:MM"
"HH:MM:SS"

wait *number_of_units* time_unit

where time_unit is one of the following:

second	**seconds**	**minute**
minutes	**hour**	**hours**

Notes

The time can be specified as a time of day (use 24-hour military clock), or as a number of seconds, minutes, or hours. If no timespec is given, the number of seconds specified in the timeout configuration setting is used.

```
wait until "12:01"
do script "backup"

; here, the time is stored in a variable
backuptime# = "01:30"
wait until backuptime#
do script "backup"

; or a delay can be specified
sendline "hello manager.sys"
wait 10 seconds
sendline "password"
```

when dde data

Defines a handler script to be executed when a DDE DATA message from a server application is received. These messages are sent each time there is a new value for an item for which a DDE ADVISE command has been issued. The script remains loaded in memory so that subsequent messages can be handled quickly. Available with WIndows and NewWave versions only.

```
when DDE data for conversation_number [item
item_name] do script [object] script_name

when DDE data for conversation_number [item
item_name] cancel
```

Notes

If an item name is included, the specified script is executed when data is received in response to a DDE ADVISE for that item. If the item name is absent, the specified script becomes a default, executed when data is received for any item which has no other script associated with it.

The handler script may use the `ddeData()` and `ddeItem()` functions to obtain the new data passed by the server and, if required, to check the item name to which the data belongs. This could be necessary if the same script is specified for several items.

To cancel the handler associated with a specific item, specify the **cancel** keyword as shown. If data is received in response to a DDE ADVISE for this item after the script associated with it has been cancelled, any default script you have established with the general form of this command is executed. To cancel the default script, specify the **cancel** keyword with no item name.

If the script is a NewWave object, the keyword **object** must be specified.

See Chapter 5 for DDE examples.

when dde execute

Defines a handler script to be executed when a DDE EXECUTE message to the Script topic is received. The script remains loaded in memory so that subsequent EXECUTE messages can be handled quickly. Available with WIndows and NewWave versions only.

```
when DDE execute do script [object]
script_name

when DDE execute cancel
```

Notes

By defining a handler procedure to intercept EXECUTE requests to the Script topic, you can save variables or settings as they were before the client's command is performed, and restore them afterwards if necessary. You could also check the contents of the command string from the client for illegal actions, or implement its own interpreter. In this case, you would write the DDE execute command string out to a file, then execute the file with `do script`.

If a handler already exists, the original handler is canceled and replaced by the new script.

To cancel the handler associated with the EXECUTE message, specify the `cancel` keyword as shown.

If the script is a NewWave object, the keyword **`object`** must be specified.

Note that to execute script commands directly the client must send a DDE EXECUTE message to the System topic.

See Chapter 5 for DDE examples.

when dde initiate

Defines a handler script to be executed when a DDE INITIATE message to the Script topic is received. The script remains loaded in memory so that subsequent INITIATE messages can be handled quickly. Available with WIndows and NewWave versions only.

```
when DDE initiate do script [object]
script_name

when DDE initiate cancel
```

Notes

It is not necessary for a DDE server script to define a DDE INITIATE procedure in order to function correctly. This handler need only be provided if you want to monitor or limit the number or type of client applications using this copy of AdvanceLink.

If a handler already exists, the original handler is canceled and replaced by the new script.

To cancel the handler associated with the INITIATE message, specify the `cancel` keyword as shown.

If the script is a NewWave object, the keyword **`object`** must be specified.

See Chapter 5 for DDE examples.

when dde poke

Defines a handler script to be executed each time a DDE POKE message to the Script topic for a specified item is received. The script remains loaded in memory so that subsequent POKE messages can be handled quickly. Available with WIndows and NewWave versions only.

when DDE poke for item **do script** [**object**] script_name

Notes

Defines a script through which a client-supplied value is placed in an item; complements the when DDE request command. For some items, you may wish to set up both a request handler and a poke handler. A limit of 64 items can be established by both commands together.

If a handler already exists, the original handler is canceled and replaced by the new script.

To cancel the handler associated with the POKE message, specify the cancel keyword as shown.

If the script is a NewWave object, the keyword object must be specified.

See Chapter 5 for DDE examples.

when dde request

Defines a handler script to be executed each time a DDE REQUEST message to the Script topic for a specified item is received. The script remains loaded in memory so that subsequent REQUEST messages can be handled quickly. Available with WIndows and NewWave versions only.

```
when DDE request for item do script [object]
script_name

when DDE request cancel
```

Notes

This is the most common command for DDE server scripts to issue. It defines the script to be executed each time the given item is requested. This command complements the `when DDE poke` command. For some items, you may wish to set up both a `request` handler and a `poke` handler. A limit of 64 items can be established by both commands together.

The handler script may use the `ddeData()` and `ddeItem()` functions to obtain the new data passed by the server and, if required, to check the item name to which the data belongs. This could be necessary if the same script is specified for several items.

If a handler already exists, the original handler is canceled and replaced by the new script.

To cancel the handler associated with the REQUEST message, specify the **`cancel`** keyword as shown.

If the script is a NewWave object, the keyword **`object`** must be specified.

See Chapter 5 for DDE examples.

when dde terminate

Defines a handler script to be executed each time a DDE TERMINATE message to the Script topic for a specified item is received. The script remains loaded in memory so that subsequent TERMINATE messages can be handled quickly. Available with WIndows and NewWave versions only.

```
when DDE terminate do script [object]
script_name

when DDE terminate cancel
```

Notes

It is not necessary for a DDE server script to define a DDE TERMINATE procedure in order to function correctly. This handler need only be provided if you want to shut down connections or print messages at termination time.

If a handler already exists, the original handler is canceled and replaced by the new script.

To cancel the handler associated with the TERMINATE message, specify the **cancel** keyword as shown.

If the script is a NewWave object, the keyword **object** must be specified.

See Chapter 5 for DDE examples.

write

Writes an expression to the write file pointer of a disk file previously opened with `open file` or `create file`.

```
write expression to file file_name
```

Notes

The file name specified must exactly match the file name specified in the `open file` command or the `create file` command. Writing always takes place at the end of the file.

```
; Open a user-specified file, write to it, and close it:
fileName#="myFile"
open file fileName#
write "any string" to file fileName#
close file fileName#
```

Related Settings

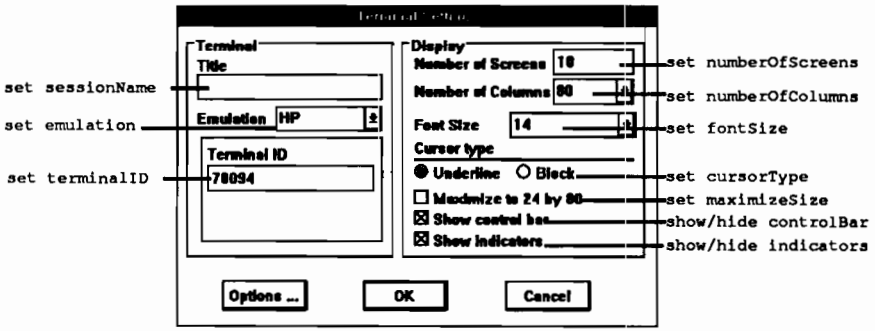
`directory`

Errors

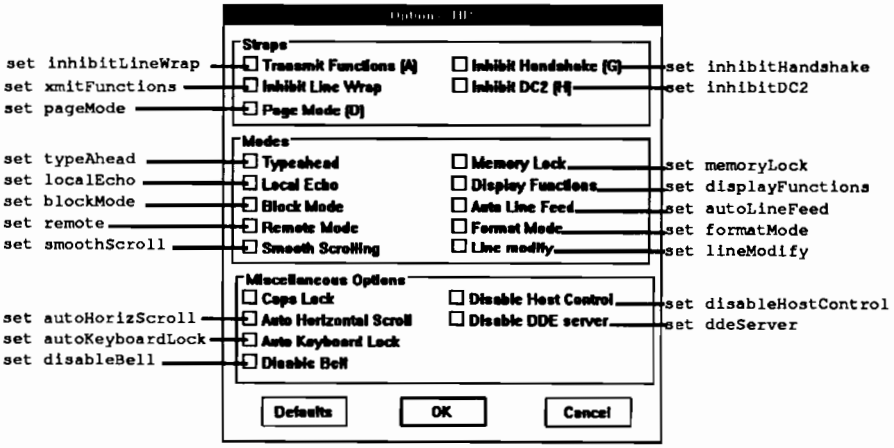
200 filePermissionError
201 fileNotFound
202 fileIOError
205 fileBusy
206 fileTooLarge
207 diskFull
208 fileNotOpen

Configuration Settings

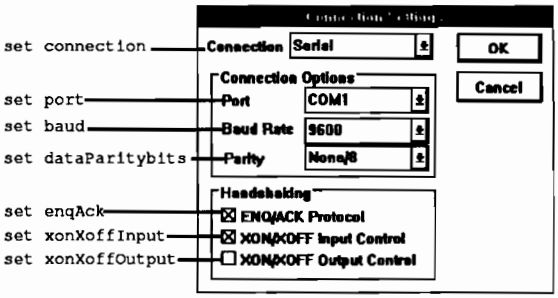
The configuration parameters listed in this section are managed with the `set` and `get` commands. The following illustrations show how the various configuration settings correspond to the options in AdvanceLink's dialog boxes and other user interface elements. Not every configuration setting corresponds to an option in the user interface, but nearly all of them do.



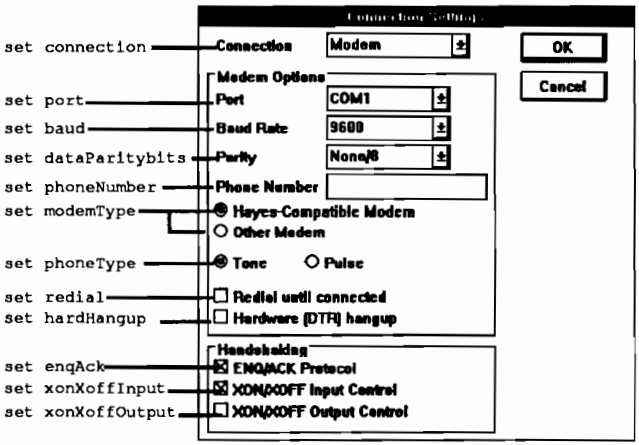
The Terminal Settings dialog box (select **Terminal** from AdvanceLink's Settings menu.)



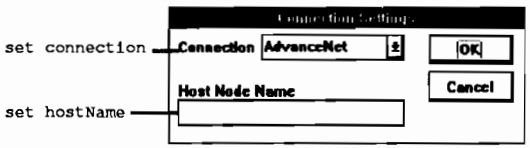
The HP Options dialog box (select **Terminal** from AdvanceLink's Settings menu; click Options button.)



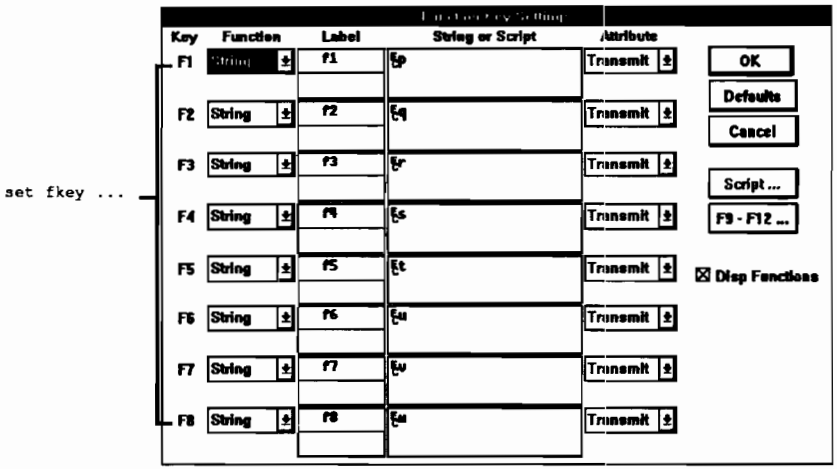
The Serial Connection Settings dialog box (select **Connection** from AdvanceLink's Settings menu; choose Serial from the Connection pop-up menu.)



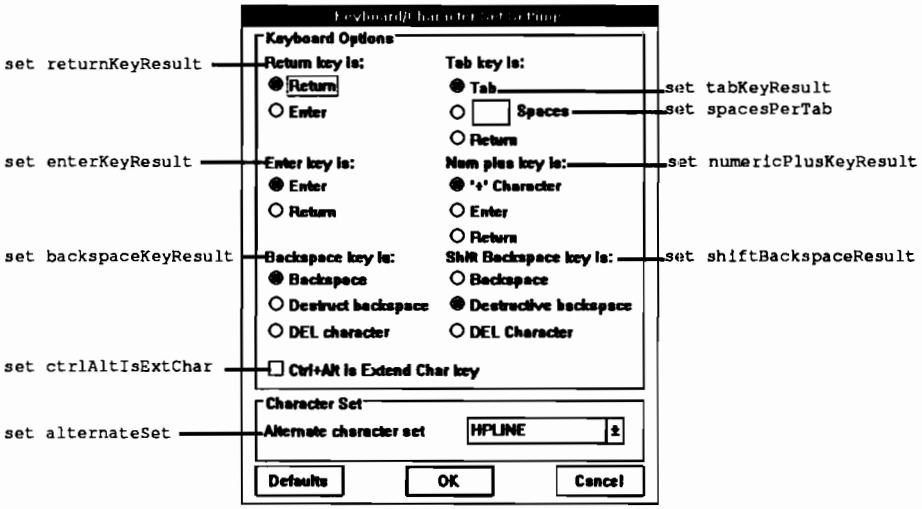
The Modem Connection Settings dialog box (select **Connection** from AdvanceLink's Settings menu; choose Modem from the Connection pop-up menu.)



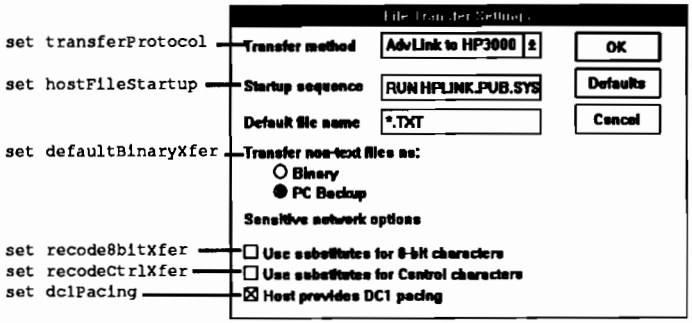
The AdvanceNet Connection Settings dialog box (select **Connection** from AdvanceLink's Settings menu; choose AdvanceNet from the Connection pop-up menu.)



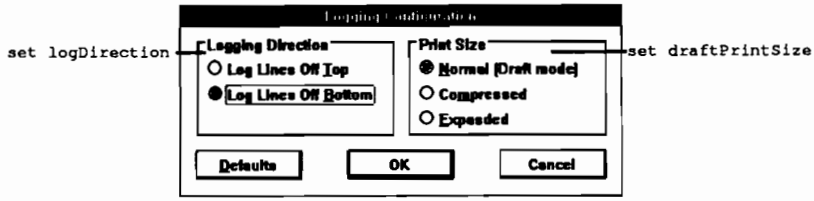
The Function Key Settings dialog box (select **Function Key** from AdvanceLink's Settings menu). Function keys are managed with special forms of the `set` and `get` commands, documented in the entries for `set` and `get` in the alphabetical commmand reference earlier in this chapter.



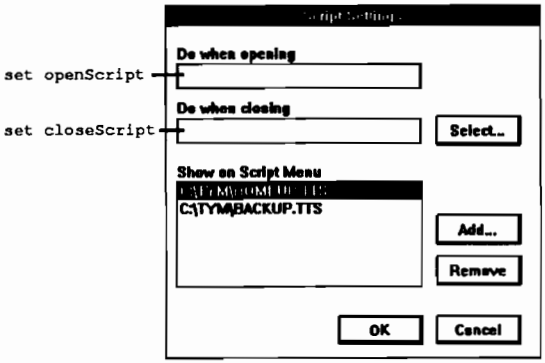
The Keyboard/Character Set Settings dialog box (select **Keyboard/Character Set** from AdvanceLink's Settings menu).



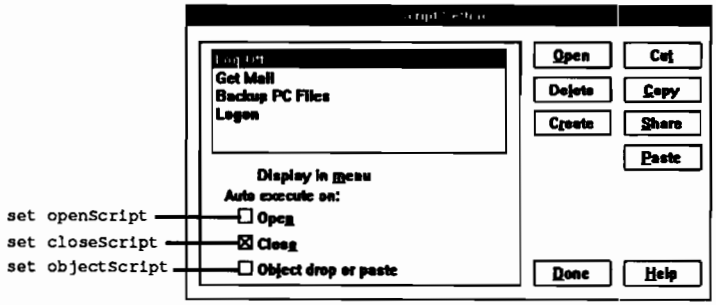
The File Transfer Settings dialog box (select **File Transfer** from AdvanceLink's Settings menu).



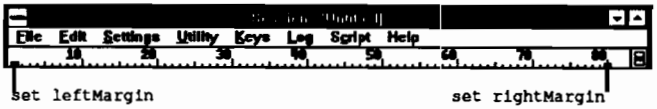
The Logging Settings dialog box (select **Logging** from AdvanceLink's Settings menu).



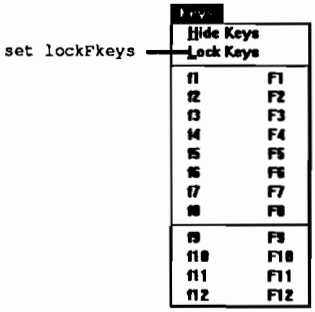
The Script Settings dialog box (select **Script** from AdvanceLink's Settings menu).



The NewWave Script Settings dialog box (select **Script** from AdvanceLink for NewWave's Settings menu).



The margin/tab ruler.



The Keys menu.

Alphabetical Listing

The parameters listed below fall into three basic types. Some have a group of predefined acceptable values, or **constants**—`logDirection`, for example, can be set to either `top` or `bottom`; `leftmargin` must be set to a number from 1 to the column number of the right margin. These are called **numeric parameters**, because even though in some cases the acceptable values are words rather than numbers, they are evaluated internally as numbers. If you attempt to set one of these parameters to a value other than those predefined, an error will result.

String parameters have no predefined values, and can contain any string which is appropriate—**`phoneNumber`**, for example, can be set to any sequence of characters you wish the modem to dial.

Boolean parameters are turned on and off—**`blockMode`**, for example, or **`keyboardLock`**. A boolean parameter must contain a numeric value, or a constant which evaluates to a numeric value. Constants include **`yes`**, **`on`**, and **`true`** (evaluate to 1) and **`no`**, **`off`** and **`false`** (evaluate to 0). Any non-zero numeric value is equivalent to **`true`**.

alternateSet

The current alternate character set. When the `emulation` parameter is set to `hpText`, `hpGraphics`, or `hpColorGraphics`, possible values are **`normal`**, **`bold`**, **`italic`**, **`linedrawing`**, and **`math`**. When the `emulation` parameter is set to `vt100` or `hpAnsi`, possible values are **`normal`**, **`lineDrawing`**, and **`uk`**.

autoHorizScroll

A boolean parameter which, when set **`on`**, enables automatic horizontal scrolling. The default value is **`off`**.

autoKeyboardLock

A boolean parameter which, when set **`on`**, enables automatic keyboard lock. The default value is **`off`**.

autoLineFeed

A boolean parameter which, when set **on**, enables the automatic linefeed option. The default value is **off**.

backspaceKeyResult

The current function of the Backspace key. Possible values are **backspaceKey**, which performs a backspace without deleting the character to the left of the cursor, **destructiveBackspace**, which performs a backspace followed by a space then another backspace, deleting the character to the left of the cursor, or **delCharacter**, which is utilized by some applications.

baud

The current baud rate. Possible values are **300**, **600**, **1200**, **2400**, **4800**, **9600**, and **19200**. The default value is 9600.

blockMode

A boolean parameter which, when set **on**, enables the block mode strap. The default value is **off**.

blocksize

The block size in bytes used during file transfer. Can be set to any value from 1 to 1024. The default blocksize value for serial and modem connections is 128 for 300 and 600 baud, 512 for 1200 baud, and 1024 for 2400 through 19200 baud rates. The default blocksize for the AdvanceNet and HP Telnet LAN connections is 1024, and the default blocksize for the BAPI LAN connection is 128. Note that if you are transferring a file from a flexible disk on the PC to a host over a 2400 baud connection, you may need to lower the blocksize to 512 to compensate for the slowness of the PC floppy disk drive.

closeScript

The name of the script performed automatically when the session file is closed.

compressXfer

A boolean parameter which, when set **on**, enables use of compression during file transfer. The default value is **off**.

connection

The current connection type. Possible values under Windows are **modem**, **serial**, **advancenet**, **telnet**, **bapi**, and **int14**. Possible values on the Macintosh are **modemPort**, **printerPort**, **advancenet**, and **telnet**.

ctrlAltIsExtChar

A boolean parameter which, when set **on**, enables the use of the Control-Alt key combination to emulate the HP Extend Char key. The Extend Char key is used in combination with other keys to produce the HP Roman-8 character set.

cursorType

The shape of the cursor. The possible values are **blockCursor** and **lineCursor**. The default value is **lineCursor**.

dataParityBits

The number of data bits used in serial datacom. Possible values are **eightNone**, **sevenEven**, **sevenOdd**, **sevenOnes**, and **sevenZeroes**. The default value is **eightNone**.

dc1Pacing

A boolean parameter which can be set **on** to enable DC1 pacing during file transfer to HP 3000 systems. The default value is **off**.

ddeApplication

The application name used by AdvanceLink when responding to DDE messages. If this parameter is not set, AdvanceLink responds to the names **Session** and **AdvanceLink**. Once another name has been set, AdvanceLink no longer responds to the default names.

Setting this parameter to a null string restores the default names.

ddeServer

The current status of AdvanceLink's ability to act as a DDE server. The possible values are **on**, **off**, and **paused**. The default value is **on**.

ddeTimeout

The default timeout period used by each of the DDE commands. The maximum value is 65 seconds; the default is 30 seconds.

ddeTemplate

The template file to be used by the Template topic. The string must be a valid MS-DOS path name.

defaultBinaryXfer

A boolean parameter which, when set **on**, enables the transfer of files in binary format. When **off**, files are transferred in backup format. The default value is **off**.

directory

Under Windows, the path name of the directory where local files are accessed. On the Macintosh, the name of a folder. The default is the directory or folder in which AdvanceLink is located.

disableBell

A boolean parameter which, when set **on**, disables the audible bell. The default value is **off**.

disableHostControl

A boolean parameter which, when set **on**, causes escape sequences for host control to be ignored. The default value is **off**.

display

A boolean parameter which, when set **off**, inhibits the display of output to the screen. The default value is **on**.

displayFunctions

A boolean parameter which, when set **on**, causes control characters to be displayed on the screen. The default value is **off**.

draftPrintSize

The current draft print size. Possible values are **normal**, **compressed**, and **expanded**. The default value is **normal**.

emulation

The current type of terminal emulation. Possible values are **hpText**, **hpGraphics**, **hpColorGraphics**, **hpAnsi**, and **vt100**.

enqAck

A boolean parameter which, when set **on**, enables HP ENQ/ACK handshaking. The default value is **on**.

enterKeyResult

The current function of the Enter key. Possible values are **returnKey** and **enterKey**.

escDelayAmount

The amount of time in milliseconds paused after an escape character is sent to the host. Possible values are from 0 to 3. The default value is 0.

fontSize

This is the size of the font used for output in the text window. Possible values are **6**, **7**, **8**, **9**, **10**, **12**, **14**, and **16**. The default value is **12**.

formatMode

A boolean parameter which, when set **on**, enables format mode. The default value is **off**.

formsCache

A boolean parameter which, when set **on**, enables local forms storage in the manner of HP 2394 & 700/94 terminals. The default value is **on**.

graphResolution

The current resolution for the graphics window. Possible values are **low**, **high**, and **hp264x**. The default value is **low**.

graphScaling

The current scaling used by the graphics window. Possible values are **none**, **fit**, and **proportional**. The default value is **none**.

graphWindows

The number of windows used to display text and graphics. Text and graphics are displayed in the same window when this parameter is set to **1**, or or in separate windows when it is set to **2**. The default value is **2**.

hardHangup

A boolean parameter which, when set **on**, lowers the DTR line when hanging up the modem. The default value is **off**.

helpFile

The name of the current help file used by AdvanceLink. This parameter is recognized only on the HP NewWave platform.

hostFileStartup

The string sent to the host to start the remote file transfer program. The default value depends on the value of the `transferProtocol` setting. If it is `link3000`, the default is `RUN HPLINK.PUB.SYS`. If it is `link9000`, the default is

`/usr/bin/hpLink`. If it is `xmodem`, no startup string is used.

hostName

The name of the host computer, used primarily for LAN connections.

hostPrompt

The prompt used by the host. The value of this parameter is dependent on the configuration of each host computer. The normal host prompt for MPE/V is ":^Q" or simply "^Q".

inhibitDc2

A boolean parameter which, when set **on**, inhibits the HP DC2 (H) strap. The default value is **off**.

inhibitHandshake

A boolean parameter which, when set **on**, inhibits the HP handshake (G) strap. The default value is **off**.

inhibitLineWrap

A boolean parameter which, when set **on**, inhibits the HP line wrap (C) strap. The default value is **off**.

keyboardLock

A boolean parameter which, when set **on**, locks the keyboard. The default value is **off**.

language

The language currently used by AdvanceLink for Macintosh. Possible values are **ascii7**, **ascii8**, **danish**, **french**, **german**, **norwegian**, **spanish**, **swedish**, **finnish**, and **uk**. The default value is **ascii7**.

leftMargin

The column position of the current left margin. Must be a number between 1 and the value of **rightMargin**. The default value is 1.

lineModify

A boolean parameter which, when set **on**, enables line modify mode. The default value is **off**.

localEcho

A boolean parameter which enables the echoing of characters typed locally. The default value is **off**.

lockFkeys

A boolean parameter which, when set **on**, locks the function keys. The default value is **off**.

logDirection

Determines whether lines are logged from the top or bottom of display memory. Possible values are **top** and **bottom**. The default value is **bottom**.

maximizeSize

Sets the size the window assumes when maximized. Possible values are **fullScreen** (the full size of the monitor) and **terminalScreen** (24 rows by 80 columns). The default value is **fullScreen**. Used under Windows only.

memoryLock

A boolean parameter which, when set **on**, enables memory lock. The default value is **off**.

modemType

The type of modem being used. Possible values are **hayes** and **nonhayes**. The default value is **hayes**.

numberOfColumns

The number of columns in display memory. Possible values are **80**, **132**, or **160**. The default value is **80**.

numberOfScreens

The number of screens in display memory. The default value is 4. The maximum number is limited by the amount of available memory.

numericPlusKeyResult

The current function of the + key. Possible values are **returnKey**, **enterKey**, and **numericPlusKey**.

objectScript

The name of the script automatically performed when a NewWave object is dragged into the AdvanceLink window, or pasted from the clipboard into AdvanceLink.

openScript

The name of the script automatically performed when the session file is opened.

pageMode

A boolean parameter which, when set **on**, enables page mode. The default value is **off**.

phoneNumber

The default phone number to be dialed by the modem.

phoneType

The type of dialing used when dialing the modem. Possible values are **tone** and **pulse**. The default value is **tone**.

port

The serial port to be used. Possible values are **com1**, **com2**, **com3**, **com4**, **port1**, **port2**, **port3**, and **port4**. The default value is **com1**.

recode8bitXfer

A boolean parameter which, when set **on**, enables the recoding of 8-bit characters during file transfer. The default value is **off**.

recodeCtrlXfer

A boolean parameter which, when set **on**, enables the recoding of control characters during file transfer. The default value is **`off`**.

redial

A boolean parameter which, when set **on**, enables automatic redialing until the modem until connected. The default value is **`off`**.

remote

A boolean parameter which, when set **on**, enables remote mode. The default value is **`off`**.

returnKeyResult

The current function of the Return key. Possible values are **`returnKey`** and **`enterKey`**.

rightMargin

The column position of the right margin. Must be a number between the value of **`leftMargin`** and the value of the **`numberOfColumns`** parameter. The default is the value of the **`numberOfColumns`** set parameter.

sessionName

The symbolic name of the current session. By default, it is the name of the open session file.

shiftBackspaceResult

The current function of the Shift-Backspace key combination. Possible values are **`backspaceKey`**, which performs a backspace without deleting the character to the left of the cursor, **`destructiveBackspace`**, which performs a backspace followed by a space then another backspace, deleting the character to the left of the cursor, or **`delCharacter`**, which is utilized by some applications.

smoothScroll

A boolean parameter which, when set **on**, enables smooth scrolling. The default value is **off**.

spacesPerTab

The number of spaces produced by the Tab key when it is **tabKeyResult** is set to **spaces**. See **tabKeyResult**.

tabKeyResult

The current function of the Tab key. Possible values are **tabKey**, **returnKey**, and **spaces**.

terminalId

The HP terminal ID string, usually `70094`, `2392A`, `2624A`, `2622A`, `2393A`, or `2397A`.

timeout

The default timeout in seconds for the **expect** command. The default value is 60 seconds.

transferProtocol

The protocol to be used for file transfers. Possible values are **link3000**, **link9000**, and **xmodem**. The default value is **link3000**.

transferTimeIncrement

The amount of time in milliseconds that AdvanceLink increases its time delay by for every packet not successfully received during an HP 3000 file transfer in which DC1 support is disabled (normally, over X.25). The default value for `transferTimeIncrement` is 100 ms. AdvanceLink begins non-DC1 file transfers using a 200 millisecond delay before sending a packet to the host. If a packet is not successfully received, the `transferTimeIncrement` is added to the delay. If successive packets are not successfully received, the `transferTimeIncrement` is again added to the delay until a packet is successfully received. In this way AdvanceLink adapts to the response time of the host system.

For networks or hosts with very slow response the `transferTimeIncrement` may need to be set to 500 or greater. This means that AdvanceLink will wait an extra 500 ms every time a packet is not successfully received by the host. For example, the delay starts at 200 ms. If the first packet is not received, AdvanceLink adds 500 ms to the delay, causing a total delay of 700 ms. If that time is still too fast and the packet is again unsuccessfully received, another 500 ms is added, making the total delay 1200 ms. This process continues until the packet is received successfully. AdvanceLink then continues to use th delay which produced a successful result for all file transfers until AdvanceLink is exited.

Normally, the default value is appropriate for most systems. The value of `transferTimeIncrement` should be kept as low as possible, since the larger the value, the slower file transfers will run.

typeAhead

A boolean parameter which, when set **on**, enables typeahead. The default value is **`off`**.

waitHostPrompt

A boolean parameter which, when set **on** causes the `sendline` and `sendstring` commands to wait for the prompt specified by the `hostPrompt` parameter before sending data. The default value is **on**.

xmitFunctions

A boolean parameter which, when set **on**, enables the transmit functions (A) strap. The default value is **`off`**.

xonXoffInput

A boolean parameter which, when set **on**, enables XON/XOFF handshaking on input from serial port. The default value is **`off`**.

xonXoffOutput

A boolean parameter which, when set **on**, enables XON/XOFF handshaking on output to serial port. The default value is **`off`**.

System Defined Functions

A **function** is a system-defined procedure which performs an operation on an **argument**, and returns a value. The argument is always specified in parentheses. AdvanceLink's functions include:

String Functions

`upper`	`lower`
`numberOfLines`	`numberOfChars`
`numberOfItems`	`numberOfWords`
`numberToString`	`StringToNum`
`find`	

Variable Functions

`isString`	`isNumber`
`isEmpty`	

System Functions

`freeDisk`	`freeMem`
`identity`	`connected`

File Functions

`newFileName`	`getFileName`
`exist`	`endOfFile`

Error Handling Functions

`error`	`errorLine`
`errorString`	

DDE Functions

`ddeConversation`	`ddeItem`
`ddeMessage`	`ddeData`

HP NewWave Functions

`objectProperties`	`currentObject`
`agentTaskRunning`	

Alphabetical Listing

In the following pages, the functions are listed in alphabetical order. A description of each function is provided. Spaces before or between the parentheses are optional.

agentTaskRunning ( )

Returns TRUE if an HP NewWave Agent Task is currently running.

```
;If the task is already running, stop
if agentTaskRunning()
   stop
;Otherwise, start the task
else
   do agent task "mytask"
endif
```

connected ( )

Returns TRUE if a successful connection has been made with the remote host.

```
if connected ()
   message prompt "Connected" icon note buttons "OK"
else
   message prompt "Not connected" icon note buttons "OK"
endif
```

currentObject ( )

Returns TRUE if there is a current HP NewWave object available for processing. A script invoked when an object is dropped into the AdvanceLink window should not usually require this function.

```
;If a current object is available, send it to the host
if currentObject ()
   send object
```

date ( )

Returns the current date in mm-dd-yyyy format.

```
display "The current date is: "+ date() + cr+ lf
```

ddeConversation( )

Returns the conversation number of the current DDE operation.

```
;Terminate the current conversation
dde terminate ddeConversation()
```

ddeData ( )

Returns data received from the most recent DDE EXECUTE, DDE POKE, or DDE ADVISE message.

```
;Get data sent by client application
mydata = ddeData()
```

ddeItem ( )

Returns the name of the item received from the most recent DDE REQUEST, DDE POKE, or DDE ADVISE message.

```
;Obtain the name of the item that the client application wishes to change
myitem = ddeItem()
```

ddeMessage ( )

Returns a constant containing the name of the last DDE message received:

`ddeinitiate` **`ddeterminate`** **`dderequest`**
`ddepoke` **`ddeexecute`**

```
message# = ddeMessage()
if message# = ddeInitiate
   display "DDE Initiated" + cr + lf
elseif
   display "DDE Terminated" + cr + lf
endif
```

endOfFile (filename)

Returns TRUE if a file opened with the `create file` or `open file` command is closed or at end of file.

```
open file "myfile"
while not endOfFile ("myfile")
   read file "myfile" into line#
   display line#
endwhile
```

error ( )

Returns 0 if no nonfatal errors have occurred in the previous command. Otherwise the error number is returned.

If an `ignore errors` command is issued so that AdvanceLink does not automatically terminate after a non-fatal error occurs, the error status is updated to reflect the success of each command after execution. The `error` function can be used to monitor this status. If no `ignore errors` command is issued, this function always returns 0.

```
ignore errors
send "myfile"
if error()
   message prompt errorString (error()) buttons "OK"
   display "Error in file: " + errorLine()
endif
```

errorLine ( )

Returns a string indicating the source file and line number of the statement that produced that last recoverable error. If no errors have been encountered, this function returns an empty string.

See example under `error()`.

errorString (error_number)

Returns a the text description corresponding to the error number specified. If the number given does not correspond to a non-fatal error then the empty string is returned. To retrieve the error message for the last error that occurred, specify `errorstring(error())`.

See example under `error()`.

exist (filename)

Returns TRUE if the specified file exists.

```
if exist (c:\autoexec.bat")
   copy file "c:\autoexec.bat" to "c:\autoexec.bak"
endif
```

find (substring, source_string, *start_character_number*)

Searches for an occurrence of a substring in a source string. If the substring is found, an integer representing its location in the source string is returned. If the substring is not found, 0 is returned.

The first expression is the substring that is the target of the search. The second expression is the source string. The third expression is the location in the source string where the search should begin. The first character in the string is considered to be position 1.

```
testData# = "abcdefgh"
foundStart# = find ("def", testData#,1)
if foundStart# <> 0
   display "Found!" + cr + lf
endif
```

freeDisk ( )

Returns the amount of free disk space (in bytes) on the currently logged drive.

```
diskSpace# = freeDisk ()
display numberToString (diskSpace#) + " bytes available on disk" + cr + lf"
memoryFree# = freeMem ()
display numberToString (memoryFree#) + " bytes free in memory" + cr + lf
```

freeMem ( )

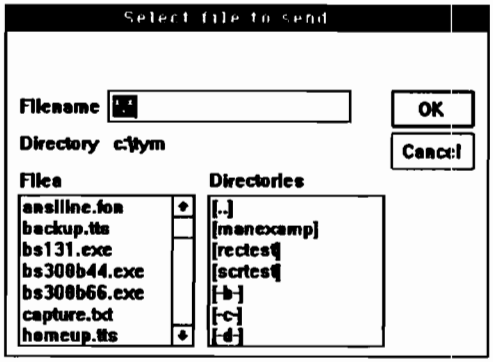
Returns the amount of free memory space (in bytes). See example under `freeDisk()`.

getFileName (prompt, qualifier)

Displays a standard Open File dialog box to allow the user to specify the name of a file to open, and returns the name of the file selected.

```
fileToSend# = getFileName ("Select file to send:","*.*")
send fileToSend# as text
```

This dialog box would be displayed:



On the Macintosh, the qualifier is a four character string that represents the file type (PICT, WORD, and so on.) Under Windows, the qualifier is a search pattern that can include MS-DOS wildcard characters. If the user selects Cancel, an empty string is returned.

Identity ( )

Returns a string representing the serial number and program version of AdvanceLink. The format of the string is as follows:

```
xxyyNVVVSSSSSSWWWtd
```

xx	Platform the emulator is running on:	
	AM	Apple Macintosh
	MW	Microsoft Windows
	WN	Windows NewWave
yy	Product number:	
	01	Business
	02	Graphics
	03	Color
VVV	Version number	
SSSSSS	Not used	
WWW	Macintosh System or Windows version number	
t	Whether TermTalk is available (Y/N)	
d	Whether DDE is available (Y/N)	

IsEmpty (variable)

Returns TRUE if the variable does not contain a value.

```
fileName# = getFileName ("Select a file to send:", "*.TXT")
if not IsEmpty (fileName#)
   send file fileName# as text
endif
```

IsNumber (variable)

Returns TRUE if the variable contains a numeric value.

```
variable1# = 10

if IsNumber (variable1#)
   display "Variable1 is a number." + cr + lf
elseif
   display "Variable1 is a string." + cr + lf
endif
```

IsString (variable)

Returns TRUE if the variable contains a string value.

```
variable1# = "10"

if IsString (variable1#)
   display "Variable1 is a number." + cr + lf
elseif
   display "Variable1 is a string." + cr + lf
endif
```

lower (expression)

Converts upper case characters in the source string to lower case and returns the resulting string. Characters that do not belong to the set A-Z are unaffected. The source string is not altered by this function.

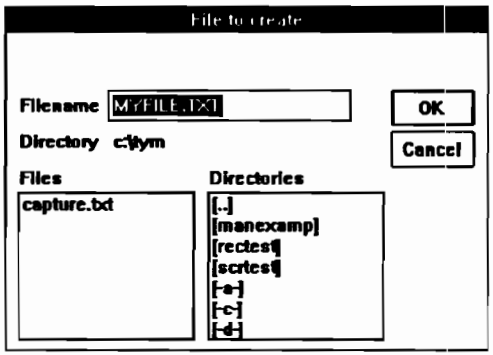
```
allower# = "abcdef"
allUpper# = "ABCDEF"

lowerResult# = lower (allUpper#)
upperResult# = upper (allLower#)

if allLower# = lowerResult# and allUpper# = upperResult#
   display "Successful!" + cr + lf
endif
```

newFileName (prompt,default_file_name)

Displays a dialog box to allow the user to specify a file to create or overwrite, and returns the file name selected. If the user clicks Cancel, an empty string is returned.

```
myFile# = newFileName ("File to create","myfile.txt")

if myfile# <> ""
   create file myfile#
endif
```

The dialog box shown below would be displayed:



numberOfChars (expression)

Returns the number of characters in the given string expression.

```
testData# = first_second_thrid" + cr + lf + "line 2" + cr + lf + "line 3,end"
display testData#
display "Data contains " numberOfChars (testData#) + "characters" + cr + lf
```

numberOfItems (expression)

Returns the number of items in the given string expression.

```
testData# = first_second_thrid" + cr + lf + "line 2" + cr + lf + "line 3,end"
display testData#
display "Data contains " numberOfItems (testData#) + "items" + cr + lf
```

numberOfLines (expression)

Returns the number of lines in the given string expression.

```
testData# = first_second_thrid" + cr + lf + "line 2" + cr + lf + "line 3,end"
display testData#
display "Data contains " numberOfLines (testData#) + "lines" + cr + lf
```

numberOfWords (expression)

Returns the number of words in the given string expression.

```
testData# = first_second_thrid" + cr + lf + "line 2" + cr + lf + "line 3,end"
display testData#
display "Data contains " numberOfWords (testData#) + "words" + cr + lf
```

numberToString (expression)

Converts the given numerical expression to a string depicting the decimal representation of the number. The contents of the string expression are evaluated until the first non-numeric character is encountered. The digits in the string are considered to be the decimal representation of a number. A null string or a string that does not begin with a digit returns a value of zero.

```
aNumber# = 10
aString# = numberToString (aNumber#)
```

objectProperties ( )

Returns a string supplying information about the properties of the current HP NewWave object. These properties are formatted one per line and can be accessed individually by using chunking expressions. The properties returned are, in order, name, type, creator, last writer, creation date, modify date, and comments. If no current object exists, this function returns error 500, noCurrentObject.

```
if line 2 of objectProperties() = "Text Note"
   send object to "tempfile"
else
   message prompt line 1 of objectProperties() &
         + " is not a Text Note" icon warn buttons "OK"
endif
```

stringToNumber (expression)

Converts the given string to a numerical value. The contents of the string expression are evaluated until the first non-numeric character is encountered. The digits in the string are considered to be the decimal representation of a number. A null string or a string that does not begin with a digit will return a value of zero.

```
aString# = "123"
aNumber# = 123
if stringToNumber (aString#) = aNumber#
   display "Equal!" + cr + lf
endif
```

time ( )

Returns the current time in HH:MM:SS format, based on a 24-hour clock.

```
display "The time is " + time() + cr + lf
```

upper (expression)

Converts all lower case characters in the source string to upper case and returns the resulting string. All characters not belonging to the set a-z are unaffected. The source string is not altered by this function.

```
allower# = "abcdef"
allUpper# = "ABCDEF"

lowerResult# = lower (allUpper#)
upperResult# = upper (allLower#)

if allLower# = lowerResult# and allUpper# = upperResult#
   display "Successful!" + cr + lf
endif
```

Using Dynamic Data Exchange (DDE) 5

Introduction

Dynamic Data Exchange (DDE) is a protocol defined by Microsoft to allow the exchange of information between programs running under Microsoft Windows. Unlike data transfers using the Clipboard, DDE interactions are fully automatic, requiring no manual involvement on the part of the user. DDE interactions can be simple one-time transfers of information, or ongoing conversations in which applications send updates to each other as new data becomes available.

AdvanceLink extends the range of DDE so that you can exchange data not only with applications residing on your PC, but with applications running on host computers. AdvanceLink can provide DDE access to data on corporate mainframes, public bulletin boards and information services, and can be used to set up hot links to real-time data such as process control instrumentation, stock market updates, and scientific instruments. These capabilities are available with the Windows and NewWave versions only.

For example, DDE capabilities might be used to obtain current stock prices from a remote service such as Dow Jones or Compuserve (accessed using AdvanceLink), and place them in an Excel spreadsheet which calculates price trends. The DDE interaction is called a **conversation**, and the participants in the conversation are called the **client** and the **server**. Typically the client requests data from the server; thus in this conversation, Excel is the client and AdvanceLink is the server.

To get a stock price as described above, you would run an Excel macro set up for this purpose. The macro would use DDE commands to load AdvanceLink and initiate the DDE conversation. At startup, AdvanceLink might automatically

run a script to dial up Compuserv and log on. Excel would then send a DDE request to AdvanceLink for the stock price, and AdvanceLink would return the result to Excel. If you like, AdvanceLink can notify Excel of changes in the price as they occur.

In another scenario, AdvanceLink might be the client and Excel the server. For example, AdvanceLink might request local sales figures from an Excel spreadsheet, using them to update a central database on the host system.

Protocol Overview

DDE conversations are established by the client, which broadcasts an INITIATE message specifying the name of the application it wishes to talk to and the **topic**, or data context, of the conversation. Topic names are defined by the server application; all applications should provide a System topic as outlined by Microsoft's standard. For document-oriented applications like word processors and spreadsheets, topics usually correspond to the names of documents.

Each window which recognizes the application and topic names broadcast in the INITIATE message sends a response to the client. If more than one reply is received, the client has a choice of servers. If no application name or topic name is specified, all windows which support DDE respond. If an application name is specified without a topic name, the answering applications send a reply for each topic they support. If a topic name is specified with no application name, all applications which support that topic respond.

(The DDE QUERY message can also be used to obtain a list of currently running applications which support DDE, along with the topics they support and their **window handles**. The window handle uniquely identifies a running window. Once you have the window handle of the application you wish to talk to, an INITIATE message can be sent directly to that window.)

Once the conversation is in progress, the client may send any of these messages:

Message	Purpose
REQUEST	Specifies a data item desired by the client. If Microsoft Excel were the server, the item might be a spreadsheet range from which to extract data. If it were AdvanceLink, the item might be either a configuration setting (for example, the current baud rate) or a piece of data from the host computer.
ADVISE	Similar to a REQUEST except that a permanent link is established; the server notifies the client each time the item changes.
POKE	Sends data from the client to the server. For example, AdvanceLink could put values into the cells of a Microsoft Excel spreadsheet. Or Excel could send data to be placed in a field in a VPLUS form.
EXECUTE	Sends instructions (for example, a macro or a script fragment) to the server to be executed.
TERMINATE	Ends the conversation. A conversation can be terminated by either the client or the server.

Data Exchange Format

Data is passed between DDE clients and servers in one of many predefined formats. A predefined item in the System topic allows applications to communicate which formats they support.

Many applications define and publish their own formats to ensure that all application-specific detail can be exchanged. Excel, for example, uses CF_BIFF to exchange cell ranges

complete with interlink formulas and other intricacies. However, virtually all applications also support some simpler, more general formats. It is the most common of these, called CF_TEXT, which AdvanceLink supports.

CF_TEXT format is straightforward ANSI text. If the text contains multiple fields of information these are separated by tab characters.

AdvanceLink also supports its own "HostSequence" format, used with the HostSequence item of the System topic, which allows DDE clients to receive binary data directly from the host.

AdvanceLink's DDE Interface

This section describes how DDE clients can address AdvanceLink as a server. Application, topic, and item names recognized by AdvanceLink are discussed in detail; the way these are put together into actual DDE commands is determined by the syntax of the client application.

Application Name

The application name recognized by AdvanceLink is **AdvanceLink**. Since AdvanceLink is marketed by Tymlabs under the name Session, AdvanceLink also recognizes the name **Session**. These defaults can be overridden by using the `set` command to modify the `ddeApplication` configuration setting. This is useful in situations where multiple copies of AdvanceLink may be running simultaneously, and client applications need to distinguish among them so they can talk to the window which is running on the desired host with the appropriate configuration settings, and so on. The `ddeApplication` name is similar in function to the session file name, but since it is less accessible to the user, it is less likely to be modified without due consideration of the implications.

Topics

The topics supported by AdvanceLink are as follows:

Topic Name	Contents
`System`	Contains items predefined by Microsoft which provide general information about the application.
	Provides access to all configuration settings (parameters of TermTalk's `set` and `get` commands).
	Allows client to send data directly to the host, to control AdvanceLink's screen display and cursor movement, and to emulate the pressing of keys on the terminal keyboard.
`Memory`	Provides access to the contents of display memory.

Template	Allows client to retrieve data from formatted-screen applications through the use of templates set up to simplify this task.
Script	Allows client to retrieve data obtained by running a script. Useful where relatively complex interaction with the host is required to locate the data.
Variables	Provides access to permanent variables.

In most cases, a client's intention in making a DDE request to AdvanceLink is to retrieve some piece of information from the host. AdvanceLink's topics provide a number of different ways of accomplishing that goal. The topics are discussed in detail below.

System Topic

Standard Items

The System topic contains a set of predefined standard items, listed below. These items can be accessed by DDE clients using the REQUEST and ADVISE messages.

Item	Description
SysItems	A list of the items supported under the System topic by AdvanceLink.
Topics	A list of the topics supported by AdvanceLink: System, Memory, Template, Script, and Variables.
ReturnMessage	Provides detail on the previous DDE ACK message (an internal message sent to acknowledge receipt of a DDE command) for error checking purposes. Returns a two-character hex number matching the internal binary code returned with the previous ACK message:

Code	Meaning
00	OK
01	Item not recognized

	02	Invalid clipboard format
	03	Insufficient memory
	04	Illegal transaction
	05	Too many ADVISE requests
	06	ADVISE already established
	07	Script compile failed
	08	Could not remove ADVISE
	09	Illegal character in type string
	0A	Could not set or get item value
	0B	TermTalk interpreter failed to run
	0C	Script forced this response
	0D	Script failed to respond
	0F	Script invalid or corrupt
	10	ddeServer set to off

`Status` Three numbers, separated by tabs, which indicate the current state of the AdvanceLink application. Each character has the significance shown below.

1st digit	1 if in file transfer, 0 otherwise
2nd digit	1 if executing a script, 0 otherwise
3rd digit	1 if recording a script, 0 otherwise

`Formats` Lists the two data formats supported by AdvanceLink: "Text" and "Host Sequence."

Configuration Items

Each AdvanceLink configuration setting (that is, each parameter of the TermTalk `set` command) is a DDE item under the System topic. Client applications can access these items with DDE REQUEST and DDE POKE messages, but not the DDE ADVISE message. Depending on the particulars of the application, you could alternatively have configuration

settings made by a startup script that runs automatically when AdvanceLink is loaded, or by a script run using the Script topic. Configuration settings are listed in Chapter 4.

Host Interaction Items

The System topic provides the DDE client with the ability to interact with the host in several ways. The client can:

- Send TermTalk script commands directly to AdvanceLink using a DDE EXECUTE message.
- Send data to the host using DDE POKE messages to the **Type**, **Key**, and **Cursor** items, described below.
- Intercept all data transmitted by the host by sending a DDE ADVISE message to the **Host Sequence** item, described below.
- Monitor when the user presses a function key by sending a DDE ADVISE message to the **Function Key** item, described below.

Type Item. Text sent to the Type item may be any alphanumeric string. To include control characters, precede the alpha character with a caret (^M for a carriage return). Note that carriage returns are not appended automatically.

Key Item. The Key item accepts data strings corresponding to the parameters of TermTalk's `key` command:

```
backTab          copyScreen
enter            tab
f1 - f12         shift f1 - shift f8
lock             return
select           stop
```

Cursor Item. The Cursor item accepts data strings corresponding to the parameters of TermTalk's `cursor` command. This can be either a row and column number

(absolute or relative; when relative, negative numbers may be used), or one of the keywords shown below.

```
row_number[,column_number] [relative]
```

up	**rollRight**	**homeDown**
down	**rollLeft**	**homeUp**
left	**rollUp**	**prevPage**
right	**rollDown**	**nextPage**

Host Sequence Item. The Host Sequence item passes all data received from the host, including escape sequences and control characters, to the client. The data stream is broken down into packets related to the data received each time AdvanceLink polls the data comm device but not exceeding 256 bytes.

Data for this item is transmitted in AdvanceLink's own "HostSequence" format. Under Windows, the client must register this format using the RegisterClipboardFormat function. Each packet of data transmitted in HostSequence format has an eight-byte header. The first two bytes give the length of the packet, the next six are reserved for future use, and the binary data follows.

Since the Host Sequence item can only be accessed by applications that support the HostSequence format, it is of primary interest to programmers developing custom applications.

Function Key Item. The Function Key item notifies the client of each user function key pressed and passes the key number, label text and transmitted text associated with the key.

Note that keys pressed by the users and control sequences sent by the host may also be captured by the Template topic, which allows you to define and monitor specific "events" relevant to your purpose.

Memory Topic

The Memory topic includes items which provide DDE clients with direct access to display memory. Portions of memory can be retrieved using a DDE REQUEST command for one of the following items:

rect `upper_row,left_col,lower_row,right_col`
(Retrieves the specified rectangle.)

line `[first_line [to last_line]]`
(If no line numbers are specified, retrieves the line the cursor is on.)

selection
(Retrieves the highlighted or marked area.)

screen
(Retrieves the 24 x 80 or 24 x 132 area immediately below the topmost visible line on the screen; not affected by window sizing or horizontal scrolling.)

all
(Retrieves all of memory.)

fields
(Retrieves all block mode fields.)

field `field_number`
(Retrieves the specified field.)

Note that a request for the `all` item may result in the retrieval of very large amounts of data. If the global memory allocation required for the data transfer is not available the request is rejected.

Template Topic

While the Memory topic provides the DDE client with access to the data in display memory, it requires the client to know exactly where on the screen the desired information lies, and at what point that information will be displayed there. Further, because this topic does not support the ADVISE message, the client cannot be updated as the information changes. The Template topic provides another way of retrieving data in display memory.

The Template topic requires the use of one or more text files called "template files." These files contain definitions of

templates, or masks, which are laid over the emulator screen to identify particular areas as data items which can be requested by the client. A template file can also contain definitions of events (such as the pressing of the Enter key) upon which the client is to be advised of the value of an item.

For example, consider a stock control database on an HP 9000 minicomputer. A file could be set up containing a template for each screen form of the data base system. One template would match the summary form, another the detailed description form for an individual stock item, and so on. Each template defines the location of any number of named data items such as "Part Number," "Quantity" and so on. Once these templates have been set up by someone familiar with the host application, several different DDE client applications can simply request data for the item "Quantity" without knowing anything at all about the database. If changes are made to the forms in the database, only the template file needs to be updated.

In addition to INITIATE and TERMINATE, the Template topic supports REQUEST, ADVISE, UNADVISE and POKE messages. There are four types of items within the topic: Files, Templates, Fields, and HostReady. These are covered in detail later on.

Template Files

Template files are standard text files which contain information describing the screen forms used by an application, including the location of specific data items on those forms. The files are similar in layout to Microsoft Windows' `WIN.INI` and `SYSTEM.INI` files. Like the INI files, the template file is divided into sections introduced by bracketed titles. These sections contain lists of named items and assignments. An example is shown on the next page.

```
[Control]
Title = screen,1,1,1,80
HostReady = "^[B^Q"

[Templates]
MainMenu = "XYZ Inc. Administration System"
CustMenu = "Customer Details"
MailMenu = "Internal Mail System"

[MainMenu]
YouHaveMail = screen,23,1,23,35

[CustMenu]
CustName = screen,5,5,5,35
Phone = screen,10,5,10,22
Zipcode = screen,12,5,12,11

[MailMenu]
RoomNo = screen,8,25,8,30

[Events]
Finished = f8,enter,HostReady
Customer = f3,f4
HostReady
```

[Control]. The `[Control]` section of the template file must contain two key descriptions, `Title` and `HostReady`.

`Title` specifies a field on the screen that contains the title or other unique identifier, often the top line of the screen form. The location of the field is described according to the rules set forth under "Template Layouts" in this chapter.

`HostReady` specifies a string that AdvanceLink is to watch for in the host data stream. The transmission of this string indicates that the display of the current form is complete and the host is awaiting input. With HP 3000 VPLUS applications, for example, the Host Ready string is usually escape-B DC1, or "^[B^Q". Each time the Host Ready string is received, AdvanceLink knows that a new screen form has

been displayed, and can attempt to match it with one of the templates in the file by checking its title and comparing it to those listed in the [Templates] section.

The Host Ready string must be enclosed in double quotes as shown, and it may optionally contain control characters preceded by a caret symbol. A literal caret is represented by ~^ and a double quote by ~", as in TermTalk itself.

[Templates]. The `[Templates]` section contains a list of key names for the templates in the file, indicating the Title field value that identifies the screen form corresponding to each template. The key names must follow the rules which govern TermTalk variable names. They may include upper and lower case letters as well as numbers and the underscore character, and may be up to 31 significant characters in length. The first character must be a letter or an underscore. Do not use the names "Control", "Templates," "Events," or any other name used as a key in the template file.

The titles themselves are strings, enclosed in double quotes, and may NOT include control characters. Title strings should include only the text which is seen on the screen, not any special enhancements.

In the template file shown above, there are three key names: MainMenu, CustMenu, and MailMenu.

Template Layouts. For each key name in the `[Templates]` section, there must be a matching section with the key name as its title, enclosed in brackets. This section establishes names for data fields on that screen, and specifications which pinpoint the location of each field.

Field names follow the rules which govern TermTalk variable names. Do not use the name "Title" or any other name used as a key in the template file.

A field specification defines a rectangle, and is composed of five parameters:

```
screen | memory | cursor, upper_row,
left_col, lower_row, right_col
```

The relative position of the rectangle described depends on the first parameter.

Parameter	Meaning
`screen`	The most common type. Rectangle is described relative to the top left corner of the 24x80 terminal screen. Zeroes and negative numbers are not used in `screen` type specifications; the first column on the screen is 1 and the first row is 1.
`memory`	Rectangle is described in terms of the whole of the display memory. This may be useful with two-page forms, or when capturing a large amount of data from the host before attempting to analyse it. Zeroes and negative numbers are not used in `memory` type specifications; the first column in display memory is 1 and the first row is 1.
`cursor`	Rectangle is described in terms of the current cursor position, which is considered row 0, column 0. Thus both zeroes and negative numbers can be used when specifying cursor type fields. The `cursor` type is most frequently used when interacting with a host application which is not form-oriented, such HP Desk , where it can be used to locate user responses to character-mode prompts.

[Events]. While the Host Ready string defined in the [Control] section of the file is used to determine when the host has completed the display of a particular screen form, Events are used to determine when the user has entered

changes to the screen. Thus the DDE client can monitor both the original data displayed by the host applications and data entered by the user, depending on what is specified by the DDE ADVISE command.

Each item in the `[Events]` section specifies a list of keys, separated by commas. The keys which are grouped together function interchangeably as a signal of when to gather data in response to an ADVISE message. The list can also include HostReady itself. The name you specify for each group of keys will usually correspond either to the type of event the keys denote (as with "Finished" in the example above), or to the screen in which the keys are used (as with "Customer" in the example above). Key names follow the rules governing TermTalk variables, descrbed above. Do not use the name "Host Ready" or any other name used as a key in the template file.

Communicating with the Template Topic

The Template topic supports REQUEST, ADVISE, UNADVISE and POKE messages. The item names used with these messages are composed of two or three parameters separated by periods.

```
item_type.name.event
```

For example, a DDE client might send an ADVISE message for the following item:

```
Field.YouHaveMail.HostReady
```

This item signifies that the client wishes to be informed of the contents of the field "YouHaveMail" each time the HostReady string is received from the host.

Note that item strings are not case sensitive: youhavemail is the same as YOUhaveMail, and so on.

Item Type. An `item_type` must be specified for all Template topic items. There are four different types:

Type	Purpose
`File`	The item specifies a template file name. By sending a DDE REQUEST or a DDE POKE for this item, the client can determine which template file is in use or change to a different one.
`Template`	The item specifies a template layout within the current template file. By sending a DDE REQUEST or a DDE POKE for this item, the client can determine which template is in use or change to a different one, obtaining access to a different set of data fields. By setting up a DDE ADVISE for the Template item with Host Ready as the precipitating event, the DDE client can be informed of each screen received from the host as it is displayed.
`Field`	The item specifies a field within the current template. Sending REQUEST or ADVISE messages for this item retrieves data from the host application. Sending a POKE message for this item allows the client to place data in the specified field.
`HostReady`	The item specifies the HostReady string for the current template file.

Name. A `name` must be specified for all Template topic item types except File items. The contents of the `name` parameter varies depending on the item type. For Template items, it is a template name defined in the current file. For HostReady items, it is any string. For field items, there are three possibilities:

- A field name from the current template file. If this field occurs in more than one template, any occurrence will

match this item. Thus if the field name has been defined for several different templates, perhaps located in different positions on the screen, data can be obtained from every screen it appears in.

- A template name, followed by a colon, followed by a field name—for example, CustMenu:Phone. In this case, the client only receives data for the item if the current template matches that specifed. If the field Phone occurs in several different templates but has different meanings in each context, specifying this form of the field name can prevent the client from receiving the wrong data.

- An asterisk. When an asterisk is specified as the field name on a DDE REQUEST or ADVISE message, values for all fields in the current template are returned, separated by tabs.

The DDE client may poke data into a field or fields, thus changing the contents of display memory. This is most useful with HP block mode applications, which read all fields on the screen only after the Enter key has been pressed. It may also be used for displaying messages to the user which look as if they have been sent from the host application.

If a POKE message is sent to all fields using the field name *, values are filled in in the order the fields are declared in the template file. Multiple adjacent tabs denote missing fields, which are left unchanged in display memory.

Event. An `event` is specified only for ADVISE messages, and defines the signal upon which AdvanceLink should advise the client of the specified data items. This event may be one of the following:

Event	Returns data
`HostReady`	When AdvanceLink receives the HostReady string from the host.
`event name`	When the user presses any key listed in the [Events] section of the template file under this event name. A named event group can include HostReady.
`f1 - f12` **`Enter`** **`Return`** **`Select`**	When the specified key is pressed. (Note: One or more of these keys may not be available on the personal computer keyboard in use; AdvanceLink provides emulation through menu items and key sequences.)

Script Topic

The Script topic allows the DDE client to retrieve data from host applications through the use of TermTalk scripts. Using the TermTalk command `when dde request` or `when dde poke`, a custom item name can be set up and associated with a particular TermTalk script so that the script is run automatically when the item is requested or poked. Such a script is called a **DDE service handler**, or just a handler.

For example, a custom item IBMPRICE might be set up as follows:

```
when dde request for "IBMPRICE" do script "DOWJONES"
```

This TermTalk command would typically be included in a startup script which is run when AdvanceLink is loaded.

Later when the client sends a request for IBMPRICE, the handler script DOWJONES is run automatically, executing the commands required to access the data in question. After being called the first time, the script is maintained in memory for faster turnaround on subsequent requests.

As with Microsoft Excel, the user can disable DDE responses through a configuration setting. This prevents the host connection from being "stolen" by a DDE client. To disable DDE responses, set the `ddeServer` configuration parameter `off`:

```
set ddeServer to off
```

Variables Topic

The Variables topic allows the DDE client to work with TermTalk's permanent variables. A permanent variable is maintained in memory after the script in which it is used terminates, so that it can be accessed by subsequent scripts. This topic allows the client to pass parameters to DDE handler scripts (see Script topic, above).

The client can send a REQUEST message to the Variables topic to retrieve the value of a permanent variable. The item name must correspond to the name of the desired variable.

The POKE message can be used to put data into a permanent variable. The item name corresponds to the name of the variable, and the data is the value of that variable. If the data is string data, it must be surrounded by quotes. If it is numeric, no quotes are used and the value must be composed entirely of digits.

If a POKE message is received for a permanent variable which does not exist, a variable with that name is created, and the specified value is placed in it.

TermTalk and DDE

This section is divided into three parts: commands used when AdvanceLink is acting as a DDE client making requests of other applications, commands used to set up AdvanceLink as a DDE server and to respond to requests, and configuration settings and functions used to gather information about and to control the current conversation.

AdvanceLink as Client

The commands listed on the next two pages are used in TermTalk scripts to communicate with a DDE server application. For further details on each command, see the the alphabetical reference in Chapter 4.

Note that while the underlying DDE protocol permits multiple DDE requests to be issued simultaneously, the AdvanceLink DDE engine queues DDE actions and deals with just one at a time. When executing client commands, script execution is suspended until the server has responded.

After issuing any client command, the TermTalk script should use the `error()` function to determine whether execution was successful, and if not, what error occurred. (Any use of the `error()` function must be preceded by an `ignore errors` statement, so that AdvanceLink does not simply terminate when a non-fatal error is encountered.) One of the following values will be returned:

Value	Meaning
0	Accepted.
400	Rejected: The request is invalid or the server does not wish to respond.
401	Busy: The request is valid but can not be handled at this time, try again later.
402	Error: There is no such conversation either because of a script programming error or because the other application in the conversation has terminated it.
403	No reply to a DDE INITIATE message.

404 Timeout: The command timed out before receiving an acknowledgement. In most cases this does not mean that the command will not take effect, only that the response from the target of the message did not arrive in time.

405 (When AdvanceLink is acting as a server.) Failed to set up script handler with a "when dde..." command.

406 (When AdvanceLink is acting as a server.) The response given in the DDE respond command is illegal.

DDE client commands are listed below. They are also included in the alphabetical reference in Chapter 4.

dde query
Obtains a list of all currently-running applications which support DDE, and/or their DDE topics.
```
dde query [application name][topic name]into variab
```

dde initiate
Initiates a DDE conversation.
```
dde initiate [window window_handle] [application na
[topic name] into conversation_number
```

dde request
Requests a data item from the server .
```
dde request conversation_number item name into
variable [wait number_of_units second[s]]
```

dde advise
Establishes a permanent link between AdvanceLink and the server application such that the server notifies AdvanceLink of the value of a specified item each time the value changes.
```
dde advise conversation_number item name [wait
number_of_units second[s]]
```

dde execute
Sends a command string to be executed by the server.

```
DDE execute conversation_number command string [wait
number_of_units second[s]]
```

dde poke
Sets the value of the specified item in the server application to the result of a specified expression.

```
dde poke conversation_number item name with expression
[wait number_of_units second[s]]
```

dde unadvise
Terminates monitoring of a server data item established by the dde advise command.

```
dde unadvise conversation_number item name [wait
number_of_units second[s]]
```

when dde data
Defines a handler script to be executed when a DDE DATA message from a server application is received. These messages are sent each time there is a new value for an item for which a DDE ADVISE command has been issued.

```
when DDE data for conversation_number [item item_name]
do script [object] script_name
```

dde terminate
Closes down the specified DDE conversation.

```
dde terminate expression [wait number_of_units
second[s]]
```

AdvanceLink as Server

AdvanceLink responds to five DDE topics. The System, Memory, Template and Variables topics are supported by the AdvanceLink program code itself and do not require the use of script commands. The fifth, the Script topic, allows clients to run TermTalk scripts, and relies on the use of the script commands listed in this section.

Typically an AdvanceLink session file which is to be used as a DDE server executes a startup script when loaded. This script logs on to the target host computer, sets up the scripts which will respond to client requests (called **DDE service handlers**, or just **handlers**) and turns on DDE.

The executable object code for DDE service handler scripts remains memory resident to improve the response time to DDE requests from clients. If service handler scripts are to be executed on machines with small memory capacities, script authors might consider calling subscripts from within the DDE service script to reduce memory usage. An approximate total memory size for a script can be estimated from the size of a compiled script file.

While the underlying DDE protocol permits multiple DDE requests to be issued simultaneously, the AdvanceLink DDE engine queues incoming DDE actions and deals with just one at time. This does not prevent the DDE service handlers from issuing DDE requests to other servers: only incoming DDE requests are queued in this way.

The following commands are used in TermTalk scripts to set up set up the DDE service handlers and to respond to requests from DDE clients. For further details on each command, see the the alphabetical reference in Chapter 4.

dde respond
Sends a message to the client application indicating the status of a DDE action it has requested. In the case of DDE REQUESTs, used to return data.
dde respond accept | **reject** | **busy** | `expression`

when dde execute
Defines a handler script to be executed when a DDE EXECUTE message to the Script topic is received.
when DDE execute do script [**object**] `script_name`

when dde initiate
Defines a handler script to be executed when a DDE INITIATE message to the Script topic is received.
when DDE initiate do `script` [**object**] `script_name`

when dde poke
Defines a handler script to be executed each time a DDE POKE message to the Script topic for a specified item is received, allowing the client to supply data for that item.
on DDE poke for item **do script** [**object**] script_name

when dde request
The most common DDE server command. Defines a handler script to be executed each time a DDE REQUEST message to the Script topic for a specified item is received, allowing the client to retrieve data for that item.
when DDE request for item **do script** [**object**] script_name

when dde terminate
Defines a handler script to be executed each time a DDE TERMINATE message to the Script topic is received.
when DDE terminate do script [**object**] script_name

dde terminate
Closes down the specified DDE conversation.
dde terminate expression [**wait** number_of_units **second**[**s**]]

clear dde handlers
Cancels all existing handler scripts and terminates all current DDE service conversations on the Script topic.
clear dde handlers

Configuration Settings

The following configuration parameters affect AdvanceLink's DDE operations. The set command is used to change the setting of a parameter; the get command reads it into a variable for testing.

ddeServer
Enables and disables DDE service. The default state when AdvanceLink is loaded is established by configuration settings made by the user and saved in the session file; if the user has made no changes, the default is on. When DDE service is on, any handlers defined with the when dde script commands are available. Handlers can also be added while

service is active. When ddeserver is off, no DDE requests are accepted by AdvanceLink. Setting DDE service from on to off maintains all current handlers in memory but suspends DDE service for all topics. Any currently queued or new DDE actions received for any topic are rejected until service is activated again. Setting ddeServer to off does not prevent AdvanceLink from issuing DDE commands as a client.

```
set [the] ddeServer to [on|off]
get [the] ddeServer into variable
```

ddeApplication

The application name AdvanceLink responds to when receiving DDE QUERY or DDE INITIATE messages from client applications. Once this parameter has been set to a non-null string, AdvanceLink no longer responds to the default names, AdvanceLink and AdvanceLink. Specifying a null string restores the default names. All current DDE service conversations and script handlers are terminated when the application name is changed. If the defaults are still in use, a get command will return either "AdvanceLink" or "Session" depending on whether the Hewlett-Packard or Tymlabs version of the product is in use.

```
set [the] ddeApplication to [string]
get [the] ddeApplication into variable
```

ddeTimeout

Used to override the 30-second default timeout period for DDE commands. Maximum period is 65 seconds.

```
set [the] ddeTimeout to number_of_seconds
get [the] ddeTimeout into identifier
```

ddeTemplate

Specifies the template file to be used by the Template topic. Must be a valid MS-DOS path name.

```
set [the] ddeTemplate to template_filename
get [the] ddeTemplate into variable
```

Functions

The following functions can be used to retrieve information about DDE operations.

`ddeItem()`
Where DDE handler scripts are used to service more than one item, this function establishes which item is currently being requested by the client. This function may also be used when AdvanceLink is the client, when a DDE DATA handler is receiving more than one ADVISE data item from a server.

`ddeData()`
Used by DDE POKE and EXECUTE handlers to obtain the data the client wishes to poke or execute. This function may also be used by DDE DATA handlers to obtain the ADVISE data sent by a server.

`ddeMessage()`
Where a single script is used to handle multiple DDE message types, this function establishes which action the client has requested. One of the following constants is returned:
`ddeinitiate` **`ddeterminate`** **`dderequest`** **`ddepoke`** **`ddeexecute`**

`ddeConversation()`
Used to obtain the conversation number for the current DDE operation. Used only when AdvanceLink is acting as the server and is going to to terminate the conversation using the `dde terminate` command.

DDE Sample Scripts

The following sample scripts show how DDE service can be established and used in the AdvanceLink environment.

Server Setup

A DDE setup script allowing access to HP Desk mail might look like this. Note that several of the service handlers share the same script. Such scripts may use the `get` command described below to establish which action they should perform.

This script terminates once AdvanceLink is running as a DDE server and has been iconized.

```
; Iconize AdvanceLink , log on to HP 3000 over LAN, and start up HP Desk
minimize
set hostPrompt to ":^Q"
connect "MARTHA.EPD.LC"
sendline "hello manager/elf.account/smile"
sendline "hpdesk 0 smith"

; Change the DDE application name to reflect our specific purpose
set ddeApplication to "HP Desk Link"

; Define DDE service handlers for accessing mail messages
when DDE initiate do script "deskinit"
when DDE terminate do script "deskterm"
when DDE poke for "TRAY" do script "desktray"
when DDE request for "READ MESSAGE" do script "deskread"
when DDE request for "MESSAGE COUNT" do script "deskcnt"
when DDE poke for "MESSAGE NUMBER" do script "deskpos"
when DDE poke for "DELETE MESAGE" do script "deskdel"
when DDE poke for "SEND MESSAGE" do script "desksend"

; Enable the DDE handlers
set ddeServer to on
```

Service Handlers

In the setup script on the preceding page, the script `deskpos` is established as the handler for DDE POKE messages to the `message number` item, and `deskread` is the handler for REQUEST messages to the `read message` item. These scripts are shown below.

```
; DESKPOS.TTS
; DDE handler for 'MESSAGE NUMBER' item for 'HP Desk Link' server
;
; The client pokes data into this item to change the current message
; number. If the supplied message number is larger than the last message in
; the tray, the message is rejected. Note that the msgCount variable is set
; to the size of the tray by the DESKTRAY.TTS script.

permanent msgCount
permanent msgNum
if stringToNumber(ddeData()) <= msgCount
   msgNum = stringToNumber(ddeData())
   DDE respond accept
else
   DDE respond reject
endif
```

```
; DESKREAD.TTS
; DDE handler for 'READ MESSAGE' item for 'HP Desk Link' server

; The client should already have set the msgNum message number to be
; read using a DDE POKE to the 'MESSAGE NUMBER' item

; The variable trayPrompt will contain the appropriate prompt string
; for the current tray, for example "In Tray>^Q" and is set by the
; DESKTRAY.TTS script in response to DDE POKEs of tray names to the
; 'TRAY' item.

permanent msgNum
sendstring "read "+msgNum
expect trayPrompt into msgText
DDE respond msgText
msgText = ""

; Note that we clear the msgText variable to free up memory once we no
; longer need the message text. Desk messages can be quite large!
```

Client Example

The following script might be used by AdvanceLink to obtain the monthly sales figures from an Excel spreadsheet and enter them into a host HP 3000 database or mail message.

```
; This DDE Client routine obtains sales total for given month from an Excel
; spreadsheet and type this into a host text file.

dde query application "Excel" into topics#
; topics# now contains a list of spread sheets open or a NULL string
; if Excel is not currently running

if topics# = ""
   ; if Excel is not running, start a copy with the desired spreadsheet
   open application "EXCEL.EXE" with "C:\EXCEL\SALES.XLS"
elseif not (topics# contains "SALES.XLS")
   ; if Excel is running but does not have a copy of SALES.XLS open
   ; send a command to Excel to cause it to open the document
   ;
   ; First start a conversation with Excel's System topic, then
   ; send and Execute request, then when completed terminate the
   ; conversation.
   dde initiate application "Excel" topic "System" into sysTopic#
   dde execute sysTopic# command "[OPEN(~"C:\\EXCEL\\SALES.XLS~")]"
   dde terminate sysTopic#
endif

; There is now a copy of Excel running with SALES.XLS open and ready
; for us to request some data from it.  First open the conversation:
dde initiate application "Excel" topic "SALES.XLS" into salesTopic#

; Now obtain the month name from Row 1 Column 3
dde request salesTopic# item "R1C3" into month#

; Now get the sales total from Row 20 Column 27
dde request salesTopic# item "R20C27" into salesTotal#

; We are finished with Excel so close down the conversation
dde terminate salesTopic#

; Now enter the text into the host application
sendline "Monthly sales total for "+month#+" was "+salesTotal#
```

System-Defined Terms

A

TermTalk defines the following terms as constants. You should ensure that none of your variable names or procedure names conflict with these constants. If a variable does have the same name as a TermTalk constant, it must be terminated with a # symbol.

ACCEPT	ADD	ADVANCENET
ADVISE	AFTER	AGENT
AGENTTASKRUNNING	ALL	ALTERNATESET
AND	APPLICATION	AS
ASCII7	ASCII8	AT
ATTRIBUTE	AUTOHORIZSCROLL	AUTOKEYBOARDLOCK
AUTOLINEFEED	BACKSPACEKEY	BACKSPACEKEYRESULT
BACKTAB	BACKUP	BAPI
BAUD	BEEP	BINARY
BLACK	BLOCKCURSOR	BLOCKMODE
BLOCKSIZE	BOLD	BOTTOM
BREAK	BUSY	BUTTONS
BYTES	CANCEL	CHANGE
CHAR	CHARACTER	CHARACTERDELAY
CHARACTERS	CHARS	CHECK
CLEAR	CLIPBOARD	CLOSE
CLOSESCRIPT	COLOR	COLUMN
COM1	COM2	COM3
COM4	COMMAND	COMPILE
COMPRESSED	COMPRESSXFER	CONNECT
CONNECTED	CONNECTION	CONTAINS
CONTROLBAR	CONTROLS	COPY
COPYSCREEN	CREATE	CTRLALTISEXTCHAR
CURRENTOBJECT	CURSOR	CURSORTYPE

CUT	DANISH	DATA
DATAPARITYBITS	DATE	DC1PACING
DDE	DDEAPPLICATION	DDECONVERSATION
DDEDATA	DDEEXECUTE	DDEINITIATE
DDEITEM	DDEMESSAGE	DDEPOKE
DDEREQUEST	DDESERVER	DDETEMPLATE
DDETERMINATE	DDETIMEOUT	DEFAULTBINARYXFER
DELCHARACTER	DELETE	DESTRUCTIVEBACKSPACE
DIAL	DIALOG	DIRECTORY
DIRNAME	DISABLE	DISABLEBELL
DISABLEENH	DISABLEHOSTCONTROL	DISCONNECT
DISPLAY	DISPLAYFUNCTIONS	DO
DOWN	DRAFTPRINTSIZE	DRAW
EICON	EIGHTNONE	ELSE
ELSEIF	EMULATION	ENDIF
ENDOFFILE	ENDPROC	ENDWHILE
ENHANCEMENTS	ENQACK	ENTER
ENTERKEY	ENTERKEYRESULT	ERROR
ERRORLINE	ERRORS	ERRORSTRING
ESCDELAYAMOUNT	EVEN	EXECUTE
EXIST	EXIT	EXPANDED
EXPECT	EXPORT	F1
F10	F11	F12
F2	F3	F4
F5	F6	F7
F8	F9	FALSE
FIELDS	FILE	FILES
FILESPECIFICATION	FIND	FINNISH
FIT	FKEYS	FONTSIZE
FOR	FORM	FORMATMODE
FORMSCACHE	FREEDISK	FREEMEM
FRENCH	FROM	FULLSCREEN
GERMAN	GET	GETFILENAME
GOTO	GRAPH	GRAPHICS
GRAPHRESOLUTION	GRAPHSCALING	GRAPHWINDOWS
HANDLERS	HARD	HARDHANGUP
HAYES	HELPFILE	HIDE

HIGH	HOMEDOWN	HOMEUP
HOSTFILESTARTUP	HOSTNAME	HOSTPROMPT
HOUR	HOURS	HP264X
HPANSI	HPCOLORGRAPHICS	HPGRAPHICS
HPTEXT	ICON	IDENTITY
IF	IGNORE	IMPORT
INDICATORS	INHIBITDC2	INHIBITHANDSHAKE
INHIBITLINEWRAP	INITIATE	INPUT
INSERT	INSTALLHP 3000	INSTALLHP 9000
INT14	INTERACTIVE	INTO
ISEMPTY	ISNUMBER	ISSTRING
ITALIC	ITEM	KEY
KEYBOARD	KEYBOARDLOCK	LABEL
LANGUAGE	LEFT	LEFTMARGIN
LENGTH	LINE	LINECURSOR
LINEDRAWING	LINEMODIFY	LINEMODIFYCOLUMN
LINES	LINK3000	LINK9000
LIST	LOCALATTRIBUTE	LOCALECHO
LOCATION	LOCK	LOCKFKEYS
LOG	LOGDIRECTION	LOGGING
LOW	LOWER	MACBINARY
MAKE	MARGINS	MATH
MAXIMIZE	MAXIMIZED	MAXIMIZESIZE
MEMORYLOCK	MENU	MESSAGE
MINIMIZE	MINIMIZED	MINUTE
MINUTES	MOD	MODEM
MODEMPORT	MODEMTYPE	NAME
NEW	NEWFILENAME	NEWWAVE
NEWWAVEPACKAGE	NEXTPAGE	NO
NONE	NONHAYES	NORMAL
NORMALATTRIBUTE	NORWEGIAN	NOT
NOTE	NUMBEROFCHARS	NUMBEROFCOLUMNS
NUMBEROFITEMS	NUMBEROFLINES	NUMBEROFSCREENS
NUMBEROFWORDS	NUMBERTOSTRING	NUMERICPLUSKEY
NUMERICPLUSKEYRESULT	OBJECT	OBJECTEXISTS
OBJECTFILE	OBJECTPROPERTIES	OBJECTS
OBJECTSCRIPT	ODD	OF

OFF	ON	OPEN
OPENSCRIPT	OR	PACK
PAGE	PAGEBREAK	PAGEMODE
PAGESETUP	PAINT	PASSWORD
PASTE	PEN	PERMANENT
PF1	PF2	PF3
PF4	PF5	PF6
PF7	PF8	PHONENUMBER
PHONETYPE	POKE	PORT
PORT1	PORT2	PORT3
PORT4	PREVPAGE	PRINT
PRINTERPORT	PROC	PROMPT
PROPORTIONAL	PROTOCOL	PULSE
QUERY	QUIT	RBLINE
READ	RECEIVE	RECODE8BITXFER
RECODECTRLXFER	RECORD	RECTANGLE
REDIAL	REJECT	RELATIVE
REMOTE	REMOVE	RENAME
REQUEST	RESET	RESPOND
RESTORE	RESULT	RESUME
RETURN	RETURNKEY	RETURNKEYRESULT
REVERT	RIGHT	RIGHTMARGIN
ROLLDOWN	ROLLLEFT	ROLLRIGHT
ROLLUP	ROW	SAVE
SCALED	SCREEN	SCRIPT
SCRIPTATTRIBUTE	SCRIPTS	SCRIPTSPEED
SECOND	SECONDS	SELECT
SELECTION	SEND	SENDLINE
SENDSTRING	SERIAL	SESSION
SESSIONNAME	SET	SETUP
SEVENEVEN	SEVENODD	SEVENONES
SEVENZEROS	SHIFT	SHIFTBACKSPACERESULT
SHOW	SMOOTHSCROLL	SOFT
SPACES	SPACESPERTAB	SPANISH
START	STOP	STRINGTONUMBER
SWEDISH	TAB	TABKEY
TABKEYRESULT	TABLE	TABS

TASK	TELNET	TERMINAL
TERMINALID	TERMINALSCREEN	TERMINATE
TEXT	THE	TIME
TIMEOUT	TIMES	TITLE
TO	TONE	TOP
TOPIC	TRANSFER	TRANSFERPROTOCOL
TRANSFERTIMEINCREMENT	TRANSMITATTRIBUTE	TRAP
TRUE	TYPE	TYPEAHEAD
UK	UNADVISE	UNTIL
UP	UPPER	VT100
WAIT	WAITHOSTPROMPT	WARN
WHEN	WHILE	WINDOW
WINDOWIMAGE	WITH	WORD
WORDS	WORDWRAP	WRAP
WRITE	XMITFUNCTIONS	XMODEM
XONXOFFINPUT	XONXOFFOUTPUT	YES

Converting AdvanceLink and Reflection Command Files

B

Conversion Programs

AdvanceLink is shipped with utility programs which aid in the conversion of any AdvanceLink for DOS or Reflection command files you may have to TermTalk scripts. These programs are called `ADV2TTS.EXE` and `REF2TTS.EXE`.

Note These programs are not supported by Hewlett-Packard.

Under Windows, the conversion programs are copied automatically during installation into the same directory as AdvanceLink. Installation under NewWave is not automatic; you must manually copy the files from the installation disks. The Reflection converter requires only the file `REF2TTS.EXE`; the AdvanceLink for DOS converter requires `ADV2TTS.EXE`, `ADV2TTS.001` and `ADV2TTS.002`.

The conversion programs are run by moving to the directory where the command files are located, then typing a command like this at the DOS prompt:

```
C:ADV2TTS input_file output_file
```

or

```
C:REF2TTS input_file output_file
```

where `input_file` is the full DOS file name for the original command file, and `output_file` is the name for the

TermTalk script (use the file extension .TTS). Because neither AdvanceLink for DOS commands nor Reflection commands correspond one-to-one to TermTalk commands, not every line in your input command file can be translated by the conversion programs.

The AdvanceLink for DOS converter, ADV2TTS.EXE, takes an AdvanceLink for DOS command file and produces a TermTalk file. The old AdvanceLink commands are included in the new TermTalk file as comments. Warning messages are displayed during the conversion showing which commands the utility was unable to convert. You must review and edit converted command files before attempting to run them.

The Reflection converter, REF2TTS.EXE, takes a Reflection command file and produces a TermTalk file. Where no identical command exists, the source line is placed in the output file as a comment, along with a further comment explaining that the line could not be translated. You must review and edit converted command files before attempting to run them.

AdvanceLink for DOS Equivalents

This table lists AdvanceLink commands in the left column, and shows their TermTalk equivalents on the right.

AdvanceLink	TermTalk
7bit	`set language to ascii7`
8bit	`set language to ascii8`
add	`+`
ascii	not supported
assign	`x# = y`
attention	`sendline`
backup	not supported
baud	`set baud to 9600`

AdvanceLink	TermTalk
blocksize	`set blocksize to 128`
call	`dial "555-1212" with "ats7=200"`
chain	`do script "filename"`
chardelay	`set characterDelay`
conterm	not supported
continue	`ignore errors`
cursor	`cursor`
dclose	`close file "name"`
dcopy	`copy file "file1" to "file2"`
dscopy	`send file` and `receive file`
display	`set display to on`
div	`/`
dopen	`open file "filename"`
dpurge	`delete file "filename"`
dread	`read file "name" into variable`
drename	`rename file "oldname" to "newname"`
dtest	`exist()`
duplex	not supported
dwrite	`write source to file "filename"`
else	`else`
endif	`endif`
endlogoff	not supported
endlogon	not supported
eoftst	`if read from file "name" = empty`
execute	`do script "filename"`
exit	`exit`
expect	`expect "string"`
fastkey	`set f1 script to "script"`

AdvanceLink	TermTalk
finishfile	`set closeScript to "script"`
goto	`goto "labelname"`

handshake	`set enqack`
	`set xonxoffInput`
	`set xonxoffOutput`

hangup	`disconnect`
hex	`set recodectrlxfer to true`
hold	Variables are used to store values
hostaccess	not supported
hostcontrol	set disableHostControl
hostcopy	`set hostfilestartup to "run hplink"`
hotkey	not supported
HP 3000	`set dc1Pacing`
if	`if`
input	`input`
key	`set f1 script to "script"`
label	`label "labelname"`
length	`numberofchars()`
logfile	`start logging`
makeconfig	`save session "filename"`
mconnect	`connect` or `dial`
msg	`message`
mul	*
netchars	not supported
netchlen	`set recode8bitxfer to on`
netmode	not supported
nolog	not supported
offkey	not supported
parity	`set parity to eightNone`

AdvanceLink	TermTalk
pause	`wait`
personality	`set emulation to hptext`
port	`set connection to modem`
position	`find()`
quicksend	`sendstring "string"`
qinput	not supported
rcursor	`get row`
	`get column`
readdc	`expect`
remote	not supported
restore	not supported
retry	not supported
return	`return`
send	`sendline "string"`
sendbreak	`break`
sendfile	not supported
sinput	`input`
sub	-
tab	`tab`
terminator	`set hostprompt to ":"`
timeout	`set timeOut`
trim	not supported
type	`display file "filename"`
upper	`upper()`
use	`open session "filename"`
vconnect	`connect`
verify	not supported

AdvanceLink	TermTalk
vlogon	not supported
vname	`set hostname to "node70"`
waitclock	`wait until "00:01:25"`
waitdc	`expect`
wtimeout	`set timeout`
xreceive	`receive "localfile" from "hostfile" protocol xmodem`
xsend	`send "source" to "dest" protocol xmodem`

Reflection Equivalents

The following table lists Reflection commands in the left column, and shows their TermTalk equivalents on the right.

Reflection	TermTalk
ACCEPT	`input`
ALERT	not supported
BACKGROUND	`minimize`
BACKUP	not supported
BREAK	`break`
CHAIN	`do script; stop`
CHDIR	`get | set directory; display`
CLOSE log device	`logDest# = ""`
CLOSE file	`close file`
COLUMN	`get column; display`
;COMMENT	`; comment`
CONTINUE	`ignore errors`

Reflection	TermTalk
COPY	`copy file`
$DATE	`date()`
DCD	not supported
DIR	`list files`
$DIR	`get directory`
DISCONNECT	`disconnect`
DISPLAY	`display`
<drive>:	`set directory`
ENTER	`key enter`
ENV	not supported
EOF	`endOfFile()`
ERASE	`delete file`
ERROR	`error()`
ERROR-CODE	`error()`
EXIST	`exist()`
EXIT	`exit`
FGD	not supported
$FILE	not supported
FIND	`find()`
FOR	`list files`, plus loop with chunking
FOREGROUND	`restore`
FOUND	not directly supported
$FULLFILE	not supported
GOSUB	`do`
GOTO	`goto`
HARDEXIT	`exit`
HELP	not supported
HOLD	`expect interactive`
IF, ELSE, ENDIF	`if`, `else`, `endif`
INVOKE	`do script`

Reflection	TermTalk
KBYE	not supported
KFINISH	not supported
KGET	not supported
KRECEIVE	not supported
KSEND	not supported
:<label>	`label`
LENGTH	`numberOfChars()`
LOAD	`open session`
LOG [OFF]	`start|stop logging`
LOGPLOT	not supported
LOWER	`lower()`
MATCH	not supported
MID	use chunking
MSAVE	`save text, print file`
ON EXIT	`set closeScript`
OPEN log device	`start logging`
OPEN <filespec>	`open file`
PACK	`pack()`
PLOT	`not supported`
PRINT	`print file`
QUIET	not supported
READ	`read file`
READHOST	`expect`
RECALL	not supported
RECEIVE	`receive`
RENAME	`rename file`
RESETCOMM	not supported
RESTORE	not supported
RETURN	`return`
ROW	`get row, display`
SAVE	`save session`

Reflection	TermTalk
SCOLUMN	`get column`
SEND	`send`
$SERIAL	`serial()`
SET	`set`
SHELL	not supported
SHUTDOWN	not supported
SPEAK	not supported
SROW	`get screen row`
STOP	`stop`
STORE	not supported
$TIME	`time()`
TRANSFER	`sendline`
TRANSMIT	`sendstring`
TSR	not supported
TYPE	`open file, read file, display`
UPPER	`upper()`
VALUE	`get`
VERIFY	not supported
WAIT	`expect` or `wait`
WRITE	`write`
XRECEIVE	`receive`

Host Control

C

Executing Scripts and Commands

AdvanceLink allows a host computer to execute an entire script or almost any individual TermTalk command using the host control facility.

The following escape sequence executes the script specified, which must be compiled previously either by using the `compile` command or by executing the script while running AdvanceLink interactively.

```
ESC & o x <compiled script file name> <CR>
```

To compile and execute a TermTalk object under NewWave, the following sequence is used.

```
ESC & o X <script object name> <CR>
```

To execute an individual command, you send the text of the command to AdvanceLink in an escape sequence. AdvanceLink then compiles and executes the command. The following escape sequence executes the command specified, but does not return command completion status to the host program. The host must assume that the command was performed successfully.

```
ESC & o C <command> <CR>
```

The following escape sequence executes the command specified, and returns completion status to the host program.

```
ESC & o B <command> <CR>
```

AdvanceLink indicates success or failure of the command by returning the standard HP terminal completion codes of `S` (for successful) or `F` (for failed) followed by a carriage return.

Failure to perform a command could be caused by a syntax error in the command, use of variables or restricted commands in the escape sequence, or the command could fail for its own reasons (file not found, for example). The completion codes are returned using a type 3 block transfer (with handshaking).

The text of the command must evaluate to a complete command expression. It must contain less than 256 characters. Partial commands are not allowed. The command cannot contain variable references. Procedure calls are not allowed, though `do script` commands can be used. Function calls are also not allowed, since they return a value. `if` statements and `while` loops are not allowed in host control escape sequences, since they are not contained in one line.

The following examples show how this facility can be used:

`ESC & o B set baud to 1200 <CR>`
This escape sequence sets the baud rate for the session to 1200. AdvanceLink would return an `S` character to the host program indicating successful completion.

`ESC & o B display "A host message" <CR>`
This escape sequence displays the string on the terminal screen. AdvanceLink would return an `S` character to the host program indicating successful completion.

`ESC & o x "myscript.ttx" <CR>`
This escape sequence executes the script specified. This allows the compilation of the host control string to be bypassed, directly performing the compiled script. This results in much better performance.

`ESC & o X "My Script Object" <CR>`
This escape sequence compiles and executes the NewWave script object specified.

Error Messages

Non-Fatal Errors

The following is a list of non-fatal TermTalk errors. The error number is found in the left column, followed by the error name and the error message. Please see the section titled "Error Handling" in Chapter 2 for information on how to trap and retrieve non-fatal errors.

If you receive a fatal error, call your support representative.

100 insufficientMemory
There is not enough memory to do this command.
Suggestion: If you are running several other applications, try quitting from one or more of the other applications before running the script.

101 compileUnsuccessful
The script file was not compiled successfully.
Suggestion: Correct the error in the script file that you are trying to compile.

102 invalidSetValue
An invalid value was given for this particular set parameter.
Suggestion: Check valid values for this configuration setting.

105 stringTooLong
A string supplied to the command was too long. Strings cannot be longer than *n* characters.
Suggestion: Change the string so that it is no greater than the number of characters specified in the error message. The maximum length varies depending on the use of the string.

200 filePermissionError
The file's permissions do not allow this type of access.

Suggestion: Make sure that the operation that you are performing on the file is valid. For example, you cannot copy to an existing file if it is a read only file.

201 fileNotFound
The specified file could not be found.

Suggestion: Make sure you have specified the correct directory and file name.

202 fileIOError
A file system IO error occured.

Suggestion: Make sure the file(s) specified in the command are correct, their permisssions are correct, and so on.

203 fileExists
The specified file already exists.

Suggestion: If you are trying to create a new version of an existing file, delete the existing file first.

204 tooManyOpenFiles
There are too many open files.

Suggestion: Close one or more files and try again.

205 fileBusy
This file is in use by another application.

Suggestion: Close the file from the other application and try again.

206 fileTooLarge
The file is too large.

207 diskFull
The disk is full.

Suggestion: Delete any unnecessary files from your disk and try again.

208 fileNotOpen
The specified file is not open. You must open the file first.

Suggestion: Make sure you have opened the file before trying to read, write, or close it.

209 fileAlreadyOpen
The specified file is already open. You cannot open the same file more than once.

Suggestion: Make sure you are trying to open the correct file.

250 connectFailed
An error prevented the connection from being established.

Suggestion: Check your connection hardware and configuration parameters.

260 modemNotResponding
The modem is not responding.

Suggestion: Make sure the modem is connected and turned on. Also make sure all of the connection settings are correct.

270 expectTimeout
A timeout has occurred. The expected string was not received before specified time expired.

Suggestion: Make sure you are expecting the correct string. If so, increase the amount of time the `expect` command waits by using the `set timeout` command.

300 unableToRunLinkProgram
Unable to run the file transfer program on the host computer.

Suggestion: Select **File Transfer** from the Settings menu to verify the host startup sequence and transfer method. Make sure you are logged on to the host computer.

302 illegalConversionOption
The specified conversion option is illegal for this file type.

Suggestion: Change the conversion option specified to a valid option, or omit the "as" parameter and use the default conversion.

303 illegalRecordSpecification
The record unit specified (bytes or words) is illegal for this type of file conversion.

Suggestion: Change the record unit to the correct option or omit this parameter to use the default.

304 fileTransferCancelled
The file transfer operation was cancelled.

305 hostFilePermissionError
The permissions of the host file prevented the successful transfer of the file.

Suggestion: Make sure the host file has the correct permissions and try again.

306 fileNameTooLong
The specified file name is too long. The maximum file name length is 36 characters.

Suggestion: Make sure the host file name specified contains 36 characters or less.

307 invalidLinkVersion
The version of the link program on the host computer is invalid.

Suggestion: Make sure you are running the correct version of HPLink. Make sure the the host file startup sequence contains the group and name of the HPLink program. Check the `hosfFileStartup` configuration setting.

308 hostFileNotFound
The host file could not be found.

Suggestion: Make sure you specified the correct file name and location for host file.

309 hostFileExists
The host file already exists.

Suggestion: Add the `delete` option to your `send` command.

310 invalidRecordSize
The specified record size is invalid.

Suggestion: You cannot change the record size for some types of file formats such as `backup`. Remove the record specification and try again.

350 printError
A printer error occurred.

Suggestion: Make sure the printer is turned on and is online.

360 invalidSession
The session that was specified is invalid.

Suggestion: Make sure the session file you specified still exists and has the name you specified.

361 badConfigFile
The configuration file format is invalid.

Suggestion: Make sure the file you are loading is an AdvanceLink configuration file.

380 notAScriptFile
The file specified is not a valid compiled TermTalk script file.

Suggestion: Make sure the script file is actually a compiled TermTalk file.

390 unableToOpenApplication
The specified application could not be opened.

Suggestion: Make sure you specified the correct path and file name of an executable file.

391 errorAccessingClipboard
An error was encountered while attempting to manipulate the clipboard data.

Suggestion: The Clipboard has memory limitations. Try copying a smaller amount of data.

400 ddeRejected
The DDE transaction was rejected by the remote application.

Suggestion: Make sure the transaction that you sent is a valid transaction for the remote application.

401 ddeBusy
The remote application was busy and could not accept the DDE transaction.

Suggestion: Wait until the remote application is not busy and try sending the transaction again.

402 ddeError
A DDE error was encountered.

Suggestion: Ensure that the application you are trying to converse with is running and is monitoring DDE data.

403 ddeNoServer
Unable to find a remote application satisfying the DDE INITIATE command.

Suggestion: Make sure there is another Windows application running that supports DDE and that you have specified parameters for the `dde initiate` command which include that application.

404 ddeTimeout
A timeout occurred in a DDE command. The remote application failed to respond before the specified time expired.

Suggestion: The remote application may be busy but failed to respond with a busy message. Try sending the transaction again. If you want the command to wait longer, use the `set ddeTimeout` command.

405 ddeNoHandler
An attempt to set up a DDE handler script failed.

Suggestion: Make sure the script is a valid TermTalk script and try again.

406 ddeIllegalResponse
An invalid use of the `dde respond` command occurred.

Suggestion: The `dde respond` command accepts only `ACCEPT`, `REJECT`, `BUSY`, or a string expression. Make sure you did not specify a number.

500 noCurrentObject
There is no current NewWave object to process.

Suggestion: Make sure there is a current NewWave object being pasted or dropped.

501 invalidDOSFilename
An invalid DOS file name was specified.

Suggestion: Make sure you specified a valid DOS path and file name.

502 invalidObjectName
An invalid Object Name was specified. The name may be too long.

Suggestion: Check that the length of the object name is 32 characters or less.

503 exportFailed
Unable to export the current NewWave object to a DOS file.

Suggestion: Check the DOS file permissions, make sure there is sufficient space on the disk and that a disk is in the disk drive.

504 importFailed
Unable to import a NewWave object from the DOS file.

Suggestion: Make sure the DOS file exists. Ensure that there is sufficient space on the disk and that a disk is in the disk drive.

505 renameFailed
Unable to rename the NewWave object.

Suggestion: Check that the length of the object name is 32 characters or less.

506 hostFileNotObject
The specified host file is not a NewWave SOF Package.

Suggestion: Make sure you specified the correct host file name. On an HP 3000, a SOF file is shown with a file code of `SOF` in a `LISTF` directory listing.

507 pcFileNotObject
The specified PC file is not a NewWave SOF Package.

Suggestion: Make sure you specified the correct PC file name. You should be able to use the **Import Object** command of the NewWave Office Objects menu if this file is in the correct format.

508 agentAlreadyRunning
Unable to start an agent task because another task is already running.

Suggestion: Wait until the agent task has completed and try again. Use the `agentTaskRunning()` function to test this within a script.

509 agentTaskNotFound
Unable to locate the specified Menu Task.

Suggestion: Make sure the task you specified is listed in the Manage Menu Task dialog box, chosen from the Task menu.

510 currentObjectNotTask
The current NewWave object is not an Agent Task.

Suggestion: The object pasted, dropped, imported or received was not an Agent Task. Use the `objectProperties()` function to check the type of the object before attempting to run it as a task.

Compiler Errors

The following is a list of TermTalk compiler errors. Most of these are self-explanatory. When such an error is encountered, the compiler terminates and the appropriate message is displayed in a dialog box. If you receive an error prefixed with the label "Internal Error Cnn:," call your support representative.

'*' cannot be used with strings.

'-' cannot be used with strings.

'/' cannot be used with strings.

'<' Must be applied to operands of the same type.

'<=' Must be applied to operands of the same type.

'<>' Must be applied to operands of the same type.

'=' Must be applied to operands of the same type.

'>' Must be applied to operands of the same type.

'>=' Must be applied to operands of the same type.

'AND' cannot be used with strings.

'CONTAINS' cannot be used with numbers.

'MOD' cannot be used with strings.

'NOT' cannot be used with a string.

'OR' cannot be used with strings.

A '+' cannot be applied to a string constant.

A '-' cannot be applied to a string constant.

A button name must be a string.

A chunking range value must be a number.

A LABEL name cannot be used in an expression.

A non-terminated string was encountered. A string must have a closing quote A non-terminated string was encountered. A string must have a closing quote A GOTO statement was used to branch to '%s' which is an unknown label.

A number was found where a string was expected.

A PROCEDURE must be defined before it is called.

A PROCEDURE name cannot be used in an expression.

A script name must be a string.

A string was found where a number was expected.

A Syntax Error was encountered.

An ELSE statement is invalid here.

An ELSEIF statement is invalid here.

An ENDIF statement is invalid here.

An ENDIF statement was expected. Each IF statment must have a corresponding ENDIF statement.

An ENDWHILE statement is invalid here.

An ENDWHILE statement was expected. Each WHILE statment must have a matching ENDWHILE statement.

An error was encountered while loading the compiler's internal tables.

Suggestion: Try quitting from some other applications to free up memory and try again.

An error was encountered while opening the file named '%s'.

An identifier must represent only one VARIABLE, PROCEDURE, or LABEL. '%s' is used to represent more that one.

Chunking only applies to strings. You used chunking on a number.

Illegal use of a label.

Illegal use of a procedure.

It is invalid to assign a value to a LABEL.

It is invalid to assign a value to a PROCEDURE.

LABEL names must be unique. The LABEL name '%s' is not unique.

The 'Exist' function cannot operate on a number.

The 'Lower' function cannot operate on a number.

The 'NumberOfChars' function cannot operate on a number.

The 'NumberOfItems' function cannot operate on a number.

The 'NumberOfLines' function cannot operate on a number.

The 'NumberOfWords' function cannot operate on a number.

The 'NumberToString' function cannot operate on a string.

The 'StringToNumber' function cannot operate on a number.

The 'Upper' function cannot operate on a number.

The application name must be a string.

The check expression must be a number.

The command expression must be a string.

The compiler was aborted by the user.

The conversation specifier must be a number.

The disable expression must be a number.

The EXPECT expression must be a string.

The expression being sent to the host must be a string.

The expression following the 'WAIT UNTIL' keywords must be a string.

The expression following the 'WAIT' keyword must be a number.

The expression following the 'WITH' keyword must be a string.

The expression following the 'WITH' keyword must be a string.

The expression in a WHILE statment must evaluate to a number.

The expression in an ELSEIF statment must evaluate to a number.

The expression specifying the protocol must be a number.

The file name must be a string.

The first parameter of the 'Find' function must be a string.

The first parameter of the 'GetFileName' function must be a string.

The first parameter of the 'NewFileName' function must be a string.

The first parameter of the 'ObjectExists' function must be a string.

The item name must be a string.

The length expression must be a number.

The menu ID must be a number.

The menu name must be a string.

The name '%s', used as a LABEL has already been used to represent a VARIABLE or PROCEDURE.

The name of the new item must be a string.

The object name must be a string.

The object type must be a string.

The parameter of the 'ErrorString' Function must be a number.

The phone number must be a string.

The prompt must be a string.

The record size must be a number.

The second parameter of the 'Find' function must be a string.

The second parameter of the 'GetFileName' function must be a string.

The second parameter of the 'NewFileName' function must be a string.

The second parameter of the 'ObjectExists' function must be a string.

The session name must be a string.

The third parameter of the 'Find' function must be a number.

The topic name must be a string.

You cannot add a string and a number together.

You cannot send a number.

Index

D

E

F

T

U

V

W