

MultiSet

MultiSet is a scripting language designed for network integrators and administrators. It provides the mechanisms necessary to make applications and files transparently available to users on the network.

NetTools also includes the scripting language, Desktop Control Language (DCL). DCL, with over 200 functions and statements, allows you greater control of the Windows desktop. If you are creating new scripts, you should begin using DCL. If you are modifying existing scripts, you can continue to use MultiSet, but you should convert your MultiSet scripts to DCL scripts for compatibility in future releases of NetTools.

For information about the DCL and converting MultiSet scripts to DCL scripts, refer to your *Desktop Control Language* guide.

This chapter presents procedures to modify MultiSet scripts with the MultiSet Editor. It also describes in detail the MultiSet commands.

Scripts: Building an Integrated Menu

MultiSet scripts provide you with the features and flexibility needed to integrate Windows applications on a local area network. MultiSet is also tightly integrated with Applications Manager. By using MultiSet scripts, you can transparently manage the use of files and applications across the network within the Windows and DOS environments.

Following is a list of some of the actions MultiSet can perform through scripts:

- Attach to other file servers on your network.
- Assign a drive letter to network drives for use by your applications.
- Display important information on the Bulletin Board.
- Run or load multiple programs with a single command.
- Change or add information to your .INI files such as the WIN.INI or the SYSTEM.INI.

- Dynamically update your environment's search path for each Windows application.

MultiSet is both similar to and different from other scripting languages for Windows. It is similar in the way it allows users to perform multiple operations using a single command. Its differences, however, are more important. It was written specifically to integrate Windows applications on a network.

Many of the MultiSet commands are designed for use by network administrators and experienced end-users. However, even if you are a novice user, you may easily use some of MultiSet's more basic commands to change directories and run or load multiple programs.


Modifying Scripts

MultiSet scripts are modified using the MultiSet Editor. The MultiSet Editor is similar to Windows Notepad, but it allows you to work on more than one script at a time through its Multiple Document Interface (MDI). You can view, write, edit, copy, and print scripts in the script windows of MultiSet Editor. The editor also checks syntax, (that is, the "grammar" of your script statements) and allows you to test the script while you are editing it (Run Script).



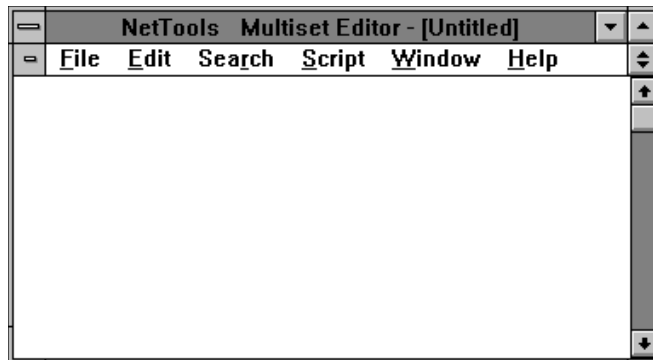
To start the MultiSet Editor

You can start the editor in one of the following ways:

-  Choose the MultiSet Editor icon if the icon exists in your Windows shell.

OR

- Choose File/ Run in File Manager or Program Manager and type MSEDIT.EXE in the Command text box.

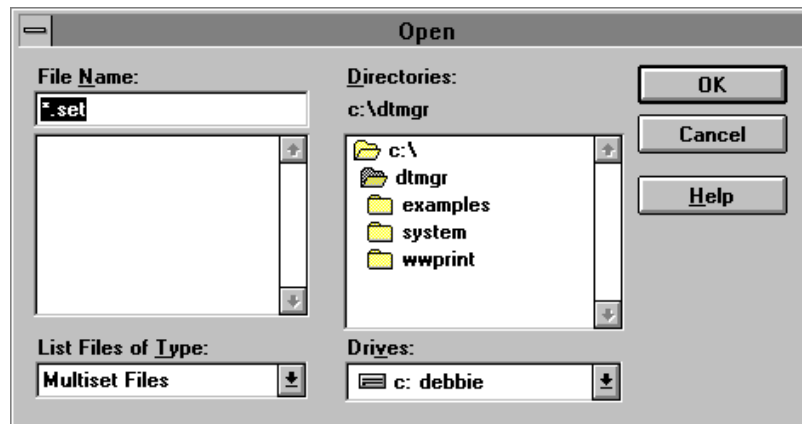


You can edit a script the same way in MultiSet Editor as in any other Windows editors. However, in addition to providing the standard Windows editing features, MultiSet Editor offers various additional editing techniques. The following sections explain each of these techniques.



To open a script file

1. Choose Open from the File menu.



2. Type or select the name of the script file (.SET) to edit in the File Name box.
3. Choose OK.

The script file displays in the MultiSet window.

Special Cursor Movement Keys

Since composing and editing scripts relies heavily upon using the keyboard, MultiSet Editor includes a number of special keys for quickly moving around in a script.

Press this...	To move to the...
CTRL+RIGHT	Next word
CTRL+LEFT	Previous word
HOME	Beginning of a line
END	End of a line
CTRL+HOME	Beginning of the script
CTRL+END	End of the script

Note: You can move the insertion point continuously by holding down any of these key combinations.

Finding Text

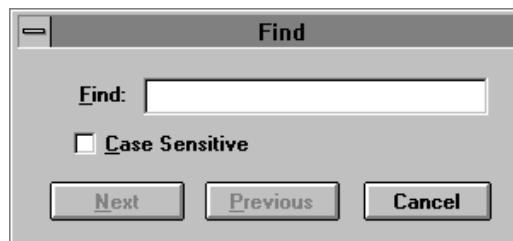
You can easily search for and edit commands in your MultiSet scripts by using the Find command on the Search menu. With this command, you can look for a character, word, or group of characters or words in a script.

When you choose the Find command, MultiSet Editor starts a search at the cursor insertion point or at the beginning of any text that is currently selected. When it reaches the end of the script, it discontinues the search and displays any appropriate messages.



To find text

1. Position the insertion point where you want to start your search.
2. Choose Find from the Search menu.



3. Type the text you want to find in the Find dialog box.
4. Select the Case Sensitive check box if you want to match capitalization exactly.
For example, if the text is *Main*, the MultiSet Editor can find *Main* but not *main*.
Otherwise, MultiSet Editor will find, for example, instances of both *Main* and *main*.

5. Choose Next (or press ENTER) to start the search in a forward direction. Choosing Previous searches in a backwards direction.

MultiSet Editor searches for the text and selects the first occurrence. If there are no occurrences of the text in the script, MultiSet Editor displays a message that the text searched for was not found.



To continue searching

1. Choose Next from the Search menu to search forward using the predefined search criteria.

Keyboard Shortcut:

Press F3.

2. Choose Previous from the Search menu to search backwards using the predefined search criteria.

Keyboard Shortcut:

Press F4.

Editing multiple scripts

With MultiSet Editor's Multiple Document Interface (MDI), it is possible to have more than one script open at a time. This feature allows you to copy parts of one script and place them in another script.

Following the standard MDI rules for Microsoft Windows, MultiSet Editor's Window menu allows you to view multiple scripts on the screen at one time. Each script is contained and edited in a window called a "child" window. You can easily manage windows with the Tile and Cascade commands on the Window menu.

You can also minimize a script to take the form of an icon on the application's workspace (background of the MultiSet Editor window). The Arrange Icons command in the Window menu allows you to arrange the icons on the application workspace. The Close All command on the Window menu allows you to close all windows simultaneously.

You can select which script, or child window, to take the focus by selecting it from the list of opened scripts in the Window menu. The scripts are numbered in the order in which they were opened. The script that is currently in focus is denoted by a check mark beside its number, path, and filename.

Checking the syntax of a script

MultiSet Editor automatically tries to help you write scripts that work properly. The Check Syntax command on the Script menu does what its name implies—checks the syntax of your script. This assures you that the command's parameters and expressions have been used properly. In addition, the editor checks your syntax every time you run a script from the editor.

As errors are discovered, a syntax error message box displays, and the cursor moves to the line and location of the invalid command.



To check the syntax of a script

1. Select the window that contains the script in which you would like to check syntax.
2. Choose Check Syntax from the Script menu.

Keyboard Shortcut:

Press CTRL+C.

Running a script from MultiSet Editor

As a convenience for testing, the MultiSet Editor's Script menu also includes a Run Script command. This command invokes MultiSet to run the script that is in the window currently in focus. If any errors are detected during the script's execution, a syntax error message displays and the cursor moves to the line and location of the offending command.



To run a script from the MultiSet Editor

1. Select the window that contains the script you wish to run.
2. Choose Run Script from the Script menu.

Keyboard Shortcut:

Press CTRL+R.

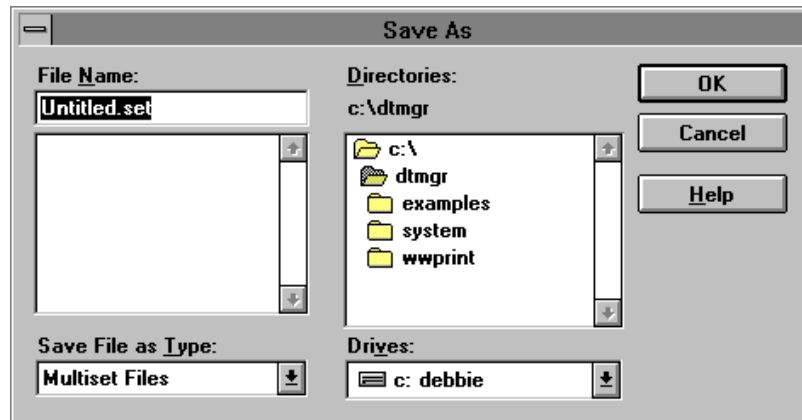
Managing MultiSet Scripts

You may Open and Save MultiSet script files much the same way you would any other text or data file—by using the commands located on the File Menu. These commands include New, Open, Save and Save As. In addition, you may also Print your scripts to the default printer. Shared MultiSet scripts should be placed in the user's search path or other accessible directory.



To save a script

1. Choose Save from the File menu. If the file has been saved previously, MultiSet saves the script file and does not display a dialog box.



2. Type the name of the script with a .SET file extension in the File Name text box. The location should be on your search path so Program Manager can find it when Windows is loaded.
3. Choose OK.
MultiSet saves the script file.

Automatically Executing Scripts

You can automatically execute MultiSet scripts (.SET files). You can use one of the following methods to execute a script:

- In Applications Manager, you can place a script file in your personal startup group (PERSTART.APP) or the network administrator can place a script file in the network startup group (STARTUP.APP). When Applications Manager loads, the script file executes.
- In Windows Program Manager, you can place a script file in the Startup group. As Program Manager loads, the script file executes.

The .SET file extension

For MultiSet to execute along with the designated .SET file, make sure that the .SET extension has been associated with the MultiSet program in the [Extensions] section of the WIN.INI file, for example:

```
SET=MULTISET.EXE ^*.SET
```

Note: MULTISET.EXE should not be included on the Run= or Load= lines of the WIN.INI file. Specifying the MultiSet executable causes a system lock-up, as the parameter for the .SET filename is never found. You can prevent this by designating only the name of the .SET file, for example EXCEL.SET.

Using MultiSet with Other Menu Systems

While MultiSet has been designed to work with Applications Manager, it works very well with any other menu system, such as Windows Program Manager.

Although Windows Program Manager was designed to be used in a stand-alone configuration, you may extend both its capabilities and its reach to the network through MultiSet. Because MultiSet handles the execution of other programs, Program Manager can run MultiSet, which, in turn, runs the application of your choice.



To place a script into Program Manager

1. Switch to Program Manager. If Program Manager is not loaded, then run PROGMAN.EXE from the File/Run menu of File Manager.
2. Select the Group Menu of your choice into which you want to place the script.
3. Choose New from the File menu.
The New Program Object dialog box displays.
4. Select Program Item and choose OK.
The Program Item Properties dialog box displays.
5. Type the description for your script in the Description text box.
6. Type the name of the script in the Command Line text box. For example, type the name WINWRITE.SET.
7. If you want to associate an icon the script, choose Change Icon. Select an icon in the Current Icon box and choose OK.
8. Choose OK to save the information in the Program Item Properties dialog box.

This adds a script to your Program Manager group menu. For more information on Windows Program Manager, refer to your documentation for Microsoft Windows.

Now, whenever you select the script from your Program Manager menu, MultiSet is in control, just as if you were running it from Applications Manager. In addition, by placing your scripts in a search path over the network, MultiSet allows you to centrally manage the way Program Manager operates, a capability which is impossible without MultiSet.

Using a script in File Manager

As another alternative, you may also run your scripts from the File Manager. By double-clicking on a .SET file, MultiSet automatically executes the selected script.

MultiSet System Variables

MultiSet allows you to change your environment and to read it. There are three commands that set your environment: PATH ADD, SET, and SETPATH. There are two ways to read your environment: with the GETPATH command and by using the environment variable enclosed in percentage signs (%) in an expression.

\$ENVMODE

When you are running Windows in standard or enhanced mode, there are two environments. One environment is used for Windows applications and one for DOS applications. To control which environment you will affect or read in a MultiSet script, you set the MultiSet system variable \$ENVMODE.

When changing the environment, there are three recognized states for this variable:

State	Description
DOS	Sets the environment for DOS applications.
Windows	Sets the environment for Windows applications. This is the default setting.
Both	Sets the environment for DOS and Windows applications.

When reading the environment, there are only two recognized states for \$ENVMODE (there is no setting for both, since the two environments could be different):

State	Description
DOS	Reads the DOS environment.
Windows	Reads the Windows environment. This is the default setting.

MultiSet expands the Windows environment as needed. After the first change is made to the Windows environment by MultiSet, there are two separate environments. Windows applications launched prior to the first environment change use the environment that was set when Windows was launched. Windows applications launched after the first environment change use the new environment which is maintained by MultiSet.

Environment Space

When making changes to the DOS environment space you may receive an “out of environment space” error message. Due to a Windows limitation, the DOS environment cannot be expanded once Windows is launched. Existing environment space, however, can be reused. You can work around this limitation by setting dummy variables in DOS to reserve the environment space before loading Windows.

You can set DUMMY to xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, for example, before launching Windows to reserve the environment space. Then reassign the environment space in the script, as in the following example which adds a directory to the path:

```
$ENVMODE=DOS
SET DUMMY=""
PATH ADD F:\WP
RUN WP.EXE
```

Restoring the Environment Setting

When you set the environment for DOS applications, MultiSet changes the environment, launches the DOS application, and restores the environment to its original state. Due to the slow speed of some systems, COMMAND.COM may be unable to finish launching before MultiSet restores the environment. To correct this occurrence, you can increase the value for the MSENVRESTORE= parameter in the [WindowsWorkstation] section of your WIN.INI file from the 1000 millisecond default setting.

\$WINVER

The variable, \$WINVER reflects the Windows version you are running. The variable will contain “30” for Windows 3.0, “31” for Windows 3.1, and “31” for Windows for Workgroups 3.11 or later.

How to Use the MultiSet Commands

MultiSet is similar to a programming language, although much easier to understand. Like most languages, it has a set of rules for putting the parts of a script together in a meaningful way. These rules are called syntax.

Usually, a MultiSet statement is made up of a command followed by its parameters. Each command expects a parameter or group of parameters. For example, the ATTACH command expects the parameter of what to attach to. Its parameters are *server* or *server/username*.

Understanding statement syntax

Scripts also need to follow certain rules for formatting commands and programming statements, although they are not case sensitive (words may be typed in upper or lower case).

The following table lists the syntax conventions used in this documentation:

Convention	Description
Commands	Words that appear in capital letters are commands. Commands must be spelled exactly as shown.
Variables	Words printed in italics or lowercase are variables. Variable information is provided by input from the user or from the results of other commands, such as MEMBEROF or LOGIN_NAME. User-defined variables are shown by \$var. User-defined variables may be used anywhere text is required.
[]	Square brackets enclose parameters that are <i>optional</i> to the command. Items enclosed within square brackets are not required in the statement, but they can be used to make a statement clearer or to add options. When you type an optional word in a command format, do not include the brackets. When two or more items may be used interchangeably, they are separated by a vertical bar ().

Continued...

Convention	Description
(. . .)	Ellipses indicate that the command or variables immediately preceding can be repeated as many times as necessary. The commands or variables must be separated by a space or other punctuation that is listed in the command's syntax.
%VARIABLE%	You may pass environment variables freely within your MultiSet scripts by using the format %VARIABLE%. For example, to pass the COMSPEC variable, use %COMSPEC%.
Other characters	Any other characters or punctuation included in a command format should be typed exactly as shown.
Special symbols	When using special symbols (! = +, etc.) in strings, the string must be enclosed in double quotes. A space is also considered a special symbol. When a section name contains a space, e.g., [Microsoft Word], enclose it in double quotes - "Microsoft Word".

Some of the commands in this reference include more than one acceptable form of syntax. In such cases, you may use any of the forms.

Command expression format

You should use the following format to compose an expression for use with MultiSet command scripts:

```
variable = expression
command
BEGIN
    command
    ...
END
IF expression THEN
    statement
ELSE
    statement
```

Words Reserved as MultiSet Commands

Words designated as MultiSet commands must be enclosed in double quotes if you want to use them as literals. For example:

```
SETINI WIN.INI WINDOWS "RUN"="WTEXT.EXE"
```

Since RUN is a MultiSet command, enclose it in double quotes to keep MultiSet from trying to execute the Run command.

WIN.INI Entries that Display in Double Quotes

Some entries in the WIN.INI are enclosed in double quotes. When indicating these strings in a MultiSet script, enclose the string in an additional set of double quotation marks to keep MultiSet from treating the text as a literal. For example, the conversion section for Microsoft Word encloses some keynames in double quotes.

```
CONV1="WordPerfect 5.1" G:\WINAPPS\WINWORD\CONV-WP5.DLL
      ^.DOC
```

When writing a script to set this line in the WIN.INI, WordPerfect must be enclosed with a double set of double quotation marks and the entire string must be enclosed in double quotation marks. Note that Microsoft Word is enclosed in double quotation marks since it contains a space.

```
SETINI WIN.INI "Microsoft Word" CONV1="""WordPerfect 5.1""
      G:\WINAPPS\WINWORD\CONV-WP5.DLL ^.DOC"
```

Combining Strings

Use the + sign to combine strings. For example:

```
CD "F:\" + LOGIN_NAME + "\sheets"
```

The MultiSet Commands

The following sections are provided as a reference to the MultiSet commands. Each command is organized into four sections. The following table describes these sections:

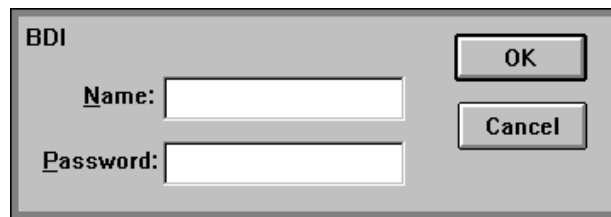
Subheading	Definition
Purpose	Provides a short description of the command and explains why you might use it.
Command Syntax	Lists the complete command format.
Returns	Shows the value that is returned for each command.
Examples	Provides examples of how the command can be used. May also provide suggestions of how to use a command in conjunction with another command.

The MultiSet commands and parameters are case insensitive. While we have used uppercase to denote the commands and lowercase to denote the parameters, this is only done for ease of reading.

ATTACH

Purpose

Checks current file server attachments. If you are already attached (as USERNAME), the command returns “TRUE”, otherwise, prompts user for *username* and *password*. The *username* can be supplied as an optional parameter. This will only prompt the user for a password.



The Attach dialog box.

Command Syntax

```
ATTACH server[/username]  
$var = ATTACH server[/username]
```

Returns

If the user is already attached to the server, the command returns “TRUE”. If the user is not attached, the command returns "" and brings up the Attach dialog box.

Example

To attach to file server MYSERVER as user SUPERVISOR, you would type the following in the MultiSet script:

```
ATTACH myserver/supervisor
```

If the user is already attached on the server specified, the user is not prompted and “TRUE” is returned.

You can also check to see if a user is currently attached to a file server by using the USERNAME command, described below. If a user is already attached and they attach again, all drive mappings to that server are lost. If any programs or data files were running from those drives, Windows may be in an unstable state. The following script is a simple example of how to check if an attachment exists and then conditionally attach to a server:

```
IF NOT (USERNAME myserver) THEN  
    ATTACH myserver/Jim
```

CD

Purpose

Changes the current working directory for MultiSet to a new directory on the same drive letter or to a new drive letter and directory. As new commands are issued or applications are launched with MultiSet, they will assume the new working directory. (This command offers enhanced functionality over the CD DOS command in that it allows you to change the current drive as well as the current directory.)

Command Syntax

```
CD [drive:]directory  
$var = CD [drive:]directory
```

Returns

If the command is successful, it returns the value of the new current directory. If the command fails, it returns "".

Example

To change to directory D:\myfiles\docs you would type:

```
CD d:\myfiles\docs
```

The CD command can be used before a LISTFILES command. This will move you to a desired directory before listing the files of your choice.

The CD command cannot contain a server/volume designation. It must use a drive letter if you want to move to a drive other than that of your current working directory. You may use the MAP command to define a new drive:\directory.

Note: If the CD command is followed by a RUN command and the application you are running uses a PIF (Program Information File) with a change directory command, the PIF change directory command overrides the MultiSet CD command.
--

COPY

Purpose

To copy a file or files from one directory, drive, or server to another.

Command Syntax

```
COPY [server\volume:][directory\]filename |
```

```
wildcard [[server\volume:][directory\]
[filename | wildcard]]
```

OR

```
COPY [drive:][directory\]filename|wildcard
[[drive:][directory\][filename|wildcard]]
$var = COPY[server\volume:][directory\]
filename | wildcard [[server\volume:]
[directory\] [filename | wildcard]]
```

OR

```
$var = COPY [drive:][directory\]filename|wildcard
[[drive:][directory\][filename|wildcard]]
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

COPY updated files to a workstation from a server.

```
COPY admin\sys2:appl\*. * c:\appl
```

```
COPY admin\sys2:batch\autoexec.bat c:\autoexec.bat
```

DELETE

Purpose

To delete a file or files from a directory, drive, or server.

Command Syntax

```
DELETE [server\volume:][directory\] filename |
wildcard
```

OR

```
DELETE [drive:][directory\]filename | wildcard
$var = DELETE[server\volume:][directory\]
filename | wildcard
```

OR


```
$var = DELETE[drive:][directory\]filename | wildcard
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

DELETE can be used to remove old files that are no longer necessary.

```
DELETE c:\pater\*.bak
```

DELINI

Purpose

Delete a section or individual parameter from a specified Windows .INI file. The .INI files must reside in the subdirectory that contains your Windows startup files in order for Windows to find it. If you do not specify an individual parameter, the entire section is deleted.

Command Syntax

```
DELINI filename.ini section [parameter]
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

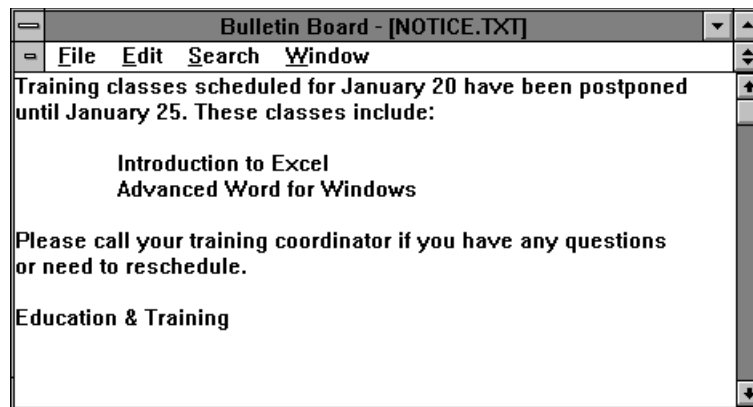
```
DELINI win.ini Extensions txt
```

This will remove the “txt=” parameter from the Extensions section of the WIN.INI file.

DISPLAY

Purpose

Displays one or more text files in a multiple document display. The first DISPLAY command in a script launches the Workstation Bulletin Board. Additional Display commands open new windows in the Bulletin Board.



Command Syntax

```
DISPLAY [server/volume:filename] | [drive:][dir\]filename
$var = DISPLAY [server/volume:filename] | [drive:][dir\]filename
```

Returns

If the command is successful, it returns "TRUE". If the command fails, it returns "".

Example

Messages or notices can be displayed for users as they login each day or as information needs to be disseminated over the LAN. One practical use might be to list the week's training classes or important events. By using the DISPLAY command with the MEMBEROF command you can dynamically present customized messages to various groups of people:

```
IF (MEMBEROF accounting) THEN
BEGIN
    MAP Q:=MYSERVER\SYS2:NEWS
    DISPLAY Q:\NEWS\ACCTNEWS.TXT
END
```

The Bulletin Board will remain on screen until closed by the user. The user can optionally copy text to the Clipboard.

EXISTS

Purpose

Determines if a particular file exists on the specified path.

Command Syntax

```
EXISTS [server\volume:]directory\filename
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

```
IF EXISTS EXCEL.EXE THEN  
    EXIT
```

In this example, the script checks the user’s current directory for the file EXCEL.EXE. If it exists, the script terminates.

EXIT

Purpose

The EXIT command terminates execution of a MultiSet script. EXIT may be used more than once in a script.

Command Syntax

```
EXIT
```

Returns

There is no return value.

Example

```
IF (USERNAME myserver) THEN  
    EXIT
```

This will exit the MultiSet script if the user is attached to file server “MYSERVER.”

FDISPLAY

Purpose

Displays a designated text file in a window with a vertical scroll bar and an OK button. Used to pop up a text file for informational purposes.

Command Syntax

```
FDISPLAY [server\volume:]directory\file
```

Returns

“TRUE” if file is displayed. Otherwise, returns “”.

Example

```
IF (MEMBEROF “admin”) THEN  
    FDISPLAY “backup.log”
```

The above example will display the file “backup.log” for a member of the group ADMIN.

GETINI

Purpose

Retrieves values from sections in the WIN.INI or other .INI files.

Command Syntax

```
$var = GETINI filename.ini section entry
```

Returns

The text in the INI file if found. Otherwise, “”.

Example

```
$var = GETINI win.ini windows spooler
```

This will get the value of spooler= from the [Windows] section of the file WIN.INI and put it in user variable \$var.

```
$shell = GETINI system.ini boot shell
```

This example allows you to determine which shell is being loaded as the user launches Windows.

GETPATH

Purpose

Gets current path information from the DOS environment. Helps in allowing a user to view or edit his path manually.

Command Syntax

```
$var = GETPATH
```

Returns

The current path setting.

Example

```
$currentpath = GETPATH  
$newpath = PROMPT "Enter New Path" $currentpath  
IF $newpath THEN  
    SETPATH $newpath
```

Displays the current PATH environment variable setting from variable \$currentpath and then prompts the user to make changes to the path. When the changes have been made, the script will update the environment with the new path. The path can also be added to the environment with the SET command.

For information on environment variables and the GETPATH command, see “MultiSet System Variables” in this chapter.

GOTO

Purpose

Jumps to *label* and continues the execution of a MultiSet script. Labels are defined by the :*label* statement.

Command Syntax

```
GOTO label
```

Returns

There is no return value.

Example

```

IF (USERNAME myserver) THEN
    GOTO showmsg
ELSE
    BEGIN
        ATTACH myserver\userid
        MAP p:=myserver\sys:userid
    END
    :showmsg
    DISPLAY myserver\sys:msgs\bbs.txt

```

If the USERNAME exists on myserver, then the script executes the line following :showmsg.

IF...THEN...ELSE

Purpose

You can use the IF...THEN command to make the MultiSet script perform commands conditionally. The expression used in evaluating the IF statement will be TRUE if the expression is anything except "". If the expression is anything more than a single operator, enclose it in parenthesis. The keywords IF and THEN must appear on the same line. You may nest IF statements within the IF-THEN-ELSE logic. BEGIN-END statements can be used to group commands to be executed inside the IF statement.

Comparison operators are used to compare user variables or identifiers. A string or variable is evaluated as True if it contains any characters. Otherwise, it is False. The following operators are available:

Operators	Description
a = b	Compares a with b and returns "TRUE" if they are equal, otherwise returns "".
a <> b	Compares a with b and returns "TRUE" if they are not equal.
NOT(a)	If a is True, returns "". If a is False, returns "TRUE".
a AND b	If a is True and b is True returns "TRUE", else returns "".

a OR b	If a is True or b is True returns “TRUE”, else returns "".
--------	--

Command Syntax

```

IF expression THEN
    command
[ELSE
    command]

```

Returns

There is no return value.

Example

```

$myname = LOGIN_NAME
IF NOT ($myname) THEN
BEGIN
    IF (MESSAGE OKCANCEL "Log in now?") THEN
    BEGIN
        IF NOT (ATTACH myserver) THEN
        BEGIN
            MESSAGE OK "Login canceled."
            GOTO fail
        END
        ELSE
            GOTO success
    END
    ELSE
    BEGIN
        MESSAGE OK "Fine."
        GOTO fail
    END
END
ELSE

```

```

        GOTO fail
:success
MESSAGE OK "Login successful."
:fail

```

The example above can determine whether a user is currently logged into the network and provide him with a mechanism for logging in.

```

IF (MEMBEROF acctsrv\accounting) THEN
    RUN NORMAL excel.exe
ELSE
    MESSAGE OK "You are not authorized to run this
               program."
EXIT

```

If the user is a member of the group “accounting” on the current server and is a user on server “acctsrv”, then a drive is mapped and a file is displayed.

ISNETWORK

Purpose

Checks to see if the network software is loaded on the current PC. May be used to conditionally run MultiSet scripts that pertain to local rather than network operation if the network is not available.

Command Syntax

```
$var = ISNETWORK
```

Returns

Returns “TRUE” if the network software has been loaded and attachments can be made, otherwise returns “”.

Example

```

IF (ISNETWORK) THEN
    GOTO connect

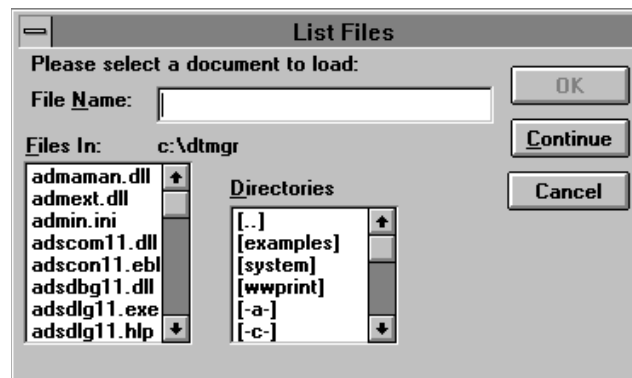
```

If the network is available, GOTO the script section labeled :connect.

LISTFILES

Purpose

Presents the user with a file selection list box using multiple file and wildcard specifications with *text*. *Filespec* is composed of multiple filespecs separated by semicolons (;) in order to display files in a particular directory for a user selection. Returns the path/filename selected or "" if the Continue button is selected. Selecting Cancel will stop the execution of the script.



Command Syntax

```
$var = LISTFILES filespec1[;filespec2...] text
```

Returns

The path/filename of the file selected, or "" if Continue is selected. If the Cancel button is selected, script execution stops.

Example

```
$file = LISTFILES *.doc;*.txt "Please select a document to load:"
```

This will display a list box of all files with .DOC and .TXT extension, and place the text "Please select a document to load:" at the top of the list box. Returns path/filename of file selected to the user variable \$file.

To display the files of a particular drive and/or subdirectory use the LISTFILES command in conjunction with a CD command like this:

```
CD "F:\\" + LOGIN_NAME + "\sheets"
```

```
$file = LISTFILES *.xl*;*.*.wk1 "Please select a spreadsheet:"
```

```
RUN NORMAL excel.exe + " " + $file
```

The CD will change the current working directory to the sheets subdirectory of the logged-in user and the LISTFILES will display the Excel and Lotus files in the directory.

As the user selects a file, he may choose to move to a different drive or subdirectory to find the file that he requires. As he moves through the subdirectories and drives, MultiSet keeps track of his location and will pass this information to the application that is launched with the file that he selects. With this feature, the user is always assured of being able to find and save his files to a familiar location from which the file came.

For example, if the user executed the script above, but decided to load a file from the N:\SHARE subdirectory, he could select the N:\SHARE drive and subdirectory from within the LISTFILES dialog box. Then when Excel is launched, it would take on that subdirectory as the current working directory. As he opened or saved files, he would see the files in the N:\SHARE subdirectory. To prevent the current working directory from changing due to a user selection in LISTFILES, you can issue a second CD command after the LISTFILES command to return the user to the proper directory before launching Excel.

LOGIN_NAME

Purpose

Returns the current username on the primary server.

Command Syntax

```
$var = LOGIN_NAME
```

Returns

Username on the primary server. This is usually the server that the user first logged in to. If "" is returned, the user is either not logged in or the network is not present. See the ISNETWORK command for information on determining whether the user has networking software loaded.

Example

```
$myname = LOGIN_NAME
```

The user's login name on the primary server will be placed in the variable \$myname. See the example in LISTFILES for another example of LOGIN_NAME.

MAP

Purpose

The MAP command can be used to map a specified drive to a given file server, volume, and directory.

Command Syntax

```
MAP [root] drive:=[server/]volume:[/directory]
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

```
MAP P:=myserver/sys:\eng\jim
```

Maps the specified drive to the given directory. Directory refers to the directory path, beginning with the volume name.

```
MAP ROOT F:=server/sys:\public\mail
```

If the optional keyword *root* is specified, the directory specified in the command will become the root directory for the drive being mapped. In the example command above, *F:/* will give access to the files in the MAIL subdirectory. You will not be able to do a change directory (CD) command on drive F: to access the files in the PUBLIC directory.

Note: Some software does not function properly when using drives with a directory mapped as the root.
--

MEMBEROF

Purpose

Determines if the user is a member of the network group called *group_name*. Groups are defined with the utilities supplied with your network operating system. For example, in Novell NetWare, you create groups through the SYSCON utility.

The server specified in this command is checked. If a server is not specified, the primary server is checked.

Command Syntax

MEMBEROF [server/] *group_name*

Returns

If the user is a member of the group, it returns “TRUE”. If the command fails, or user is not a member, returns "".

Example

IF (MEMBEROF sales) THEN

CD f:\sales\records

If the user is a member of the group sales, changes current directory and drive to f:\sales\records.

MESSAGE

Purpose

Produces a message box with buttons showing *text*. Each *type* is listed in the following table:

Type	Button(s)
OK	OK button
OKCANCEL	OK and CANCEL button
YESNO	Yes and No button
YESNOCANCEL	Yes, No, and Cancel button



A typical Message dialog box.

Command Syntax

MESSAGE *type text*

Returns

If the command is successful or the user selects OK, it returns “TRUE”. If the command fails or the user selects CANCEL, it returns “”. If the user selects Yes, the command return “YES”. If the user selects No, the command returns “NO”.

Example

```
$continue = MESSAGE YESNO “Delete selected file?”
```

This will display the text in a message box with both YES and NO buttons and set user variable \$continue with either “YES” or “NO” depending on which button is pressed.

```
MESSAGE OK “Not logged in.”
```

This will display the text “Not logged in” in a message box with just an OK button.

MKDIR

Purpose

Makes a new directory.

Command Syntax

```
MKDIR [server/volume:dir] | [drive:][dir\]
$var = MKDIR [server/volume:dir] | [drive:][dir\]
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

To create d:\myfiles\docs you would type:

```
MKDIR d:\myfiles\docs
```

PATH ADD

Purpose

Adds a drive and directory path to the global path. This new path is appended to the end of the current global path and is available to all applications that are launched through MultiSet including standard DOS applications.

PATH ADD is an important tool for the administrator as it eliminates the need for multiple drive mappings and search drive mappings to the same volume of a server. When applications are launched with underlying scripts, each script can map drives “on the fly” to ensure the proper execution of a program.

It is important to use correct path techniques to ensure that your MultiSet scripts work smoothly. The command syntax requires a complete path to the executable to ensure that an executable file can still be found if a drive mapping changes.

Command Syntax

```
PATH ADD [drive:][dir\]
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

```
PATH ADD c:\winapps\winword
```

This will add the new path c:\winapps\winword to the current global path, making the programs in that directory available to the user on the search path. This is the equivalent of MAP INSERTing a Search Drive in Novell’s NetWare or changing your path in DOS with the PATH command.

Each subsequent time that this command is run, it will check to see if you already have the requested path on your search path. If it is, it will not add it again.

Note: PATH ADD does not affect DOS boxes.
--

For information on environment variables and the PATH ADD command, see “MultiSet System Variables” in this chapter.

P_STATION

Purpose

Returns the physical station ID number of the workstation in 12 hexadecimal digit format. This number is usually stored in the network interface card either as a factory burned-in address (as it is with Ethernet) or in a definable setting (as it is with ARCNet or Token Ring). The P_STATION command is very useful for conditionally running a program like a print server based on the workstation’s hardware ID.

Command Syntax

P_STATION

Returns

The physical station ID number as text if the user is logged into the network.

Example

```
IF (P_STATION = “104225987657”) THEN
BEGIN
    RUN NORMAL pserver.exe
    EXIT
END
```

Will check the value of P_STATION first and use it in the IF statement.

PROMPT

Purpose

Prompts user for *text* by showing text and putting *default* in the edit box. If no text is in the edit box, or CANCEL is pressed, "" is returned; otherwise, the text in the edit box is returned. This command can be used to request a variable from the user during the execution of the script. The results of the command can be used by other MultiSet commands as variables.



Command Syntax

PROMPT *text default*

Returns

The text that the user entered in the edit box if OK is pressed, else "".

Example

```
$oldpath = GETPATH
$newpath = PROMPT "Please edit your path settings." $oldpath
SETPATH $newpath
```

RENAME

Purpose

To rename a file or files.

Command Syntax

RENAME *filename|wildcard filename|wildcard*


```
$var = RENAME filename|wildcard filename|wildcard
```

Returns

If the command is successful, it returns “TRUE”. If the command fails, it returns “”.

Example

RENAME can be used to change the name of a file. For example, if you keep multiple AUTOEXEC.BAT files, you might use this command.

```
RENAME autoexec.bat autoexec.net
```

```
RENAME autoexec.nm autoexec.bat
```

RIGHTS

Purpose

Determines a user’s access rights to a particular network path. If a filename is included in the path, the user’s access rights to that file are also checked.

Flag	Description
R	Read from Files
O	Open Files
S	Search Directory
C	Create New Files
D	Delete Files
M	Modify File Names/Flags
P	Parental Rights
W	Write to Files

Command Syntax

```
RIGHTS [server\volume:path\filename] [flags] | [drive:] [dir] [file]
```

Returns

If the user has at least the rights specified in *flags*, it returns “TRUE”. If the user has fewer rights, it returns “”.

Example

```
IF ($access = RIGHTS myserver\sys:excel "ROS") THEN
    RUN MAXIMIZE EXCEL.EXE
ELSE
    MESSAGE OK "Sorry, you have insufficient rights."
```

This will verify that the user has at least ROS rights in the directory myserver\sys:excel before attempting to run the program.

RMDIR

Purpose

Removes a directory.

Command Syntax

```
RMDIR [server/volume:dir] | [drive:][dir\]
$var = RMDIR [server/volume:dir] | [drive:][dir\]
```

Returns

If the command is successful, it returns "TRUE". If the command fails, it returns "".

Example

To remove d:\myfiles\docs you would type:

```
RMDIR d:\myfiles\docs
```

RUN

Purpose

Runs a program using *showtype* as the initial display mode. *Showtype* can be either "icon," "normal," or "maximize." Icon causes the application to load as an icon. Normal causes the program to run in its window that would be displayed upon a normal startup. Maximize causes the application being run to take over full screen after it has been launched.

If a variable holds the command line for a program, be sure to include a space between the EXE filename and the variable ("CLOCK.EXE" + " " + \$file).

Command Syntax

```
RUN [showtype] [path\]program [commandline]
RUN [showtype] "[path\]program [commandline]"
```

Returns

If the command is successful, it returns "TRUE". If the command fails, it returns "".

Example

```
$file = LISTFILES *.doc "Enter the file to be spellchecked."
```

```
RUN NORMAL c:\winspace\spell.exe + " " + $file
```

OR

```
RUN ICON "notepad.exe readme.txt"
```

The first example will run spell.exe from the c:\winspace directory in the normal mode using the results of the LISTFILES command as the command line argument. The second example will run Notepad, loading the file readme.txt.

SET

Purpose

Sets the DOS environment variable to a specified value.

Command Syntax

```
SET envvar = "value"
```

Returns

If the command is successful, it returns "TRUE". If the command fails, it returns "".

Example

```
SET HOME = "F:\"+LOGIN_NAME
```

For the user FRED, this will set the DOS environment variable HOME to equal F:\FRED.

For information on environment variables and the SET command, see "MultiSet System Variables" in this chapter.

SETINI

Purpose

Sets, changes, or creates a value in a section of a specified Windows .INI file. New sections can be created in existing .INI files; the .INI file must have already been created. The .INI files must reside in the subdirectory that contains your Windows startup files in order for Windows to find it.

Command Syntax

SETINI *filename.ini section parameter = value*

Returns

If the command is successful, it returns "TRUE". If the command fails, it returns "".

Example

SETINI system.ini boot shell=appman.exe

This will set the value of shell= in the [boot] section of system.ini to APPMAN.EXE.

SETPATH

Purpose

Sets the global path to *newpath*. *Newpath* can be the result of a function or a variable. This command can be used with the GETPATH command to allow the user to manually change his current path.

Command Syntax

SETPATH *newpath*

Returns

If the command is successful, it returns "TRUE". If the command fails, it returns "".

Example

\$result = SETPATH "c:\;c:\dos;c:\winword"

Sets the search path to "c:\;c:\dos;c:\winword" and returns the result to the user variable \$result. See "GETPATH" for another example of how to use SETPATH.

For information on environment variables and the SETPATH command, see “MultiSet System Variables” in this chapter.

USERNAME

Purpose

This function is useful for determining if a user is attached to a server. If attached, returns the user’s name for the requested server.

Command Syntax

```
USERNAME server
```

Returns

User’s name if attached, otherwise it returns "".

Example

```
IF NOT(USERNAME server1) THEN  
IF NOT(ATTACH (server1 + “\”+LOGIN_NAME)) THEN  
EXIT
```

The IF statement checks to see if the user is attached to server1. If the user is not attached, the script will prompt the user for a password using his current LOGIN_NAME (from his primary server) and attempt to attach to server1 using it. If this is not successful, the script is exited.

