5 Command Reference

5.1 Summary

General Functions and Statements

Beep	Statement	Plays a beep on the PC speaker
Begin	Statement	Required; Marks the beginning of the script
Clear	Statement	Clears the runtime monitor display
End	Statement	Required; Marks the end of the script
Exit	Statement	Causes Opscripts.exe to terminate when encountered
If… Then… Else	Statements	Conditional check
Loop… EndLoop	Statements	Unconditional loop
OnError	Statement	Determines how to proceed in the event an error occurs
Script	Statement	Required; Identifies this file as a script
SendOnly	Statement	Initiates an Outpost Send Only session
SendReceive	Statement	Initiates an Outpost Send/Receive session
Var	Statement	Defines a user variable
While.. EndWhile	Statements	Conditional loop
Now()	Functipon	Returns date and time based on user formatting
Pause()	Function	Causes the script to pause
Play()	Function	Plays a .wav file
Print()	Function	Prints a string of text to the Runtime Monitor window
Run()	Function	Run a program, does not wait for it to complete
Runw()	Function	Run a program, waits for it to complete

File Functions

Delete()	Function	Deletes a file
Exists()	Function	Tests if a file exists
FindFile()	Function	Sets up to find matches to a file mask
GetFileName()	Function	Returns the file name only from a full path file string
MoveFile()	Function	Moves a file to a different directory
NextFile()	Function	Gets the next file that matches a file mask
ReadFile()	Function	Read a file content
ValidFileName()	Function	Creates a valid file name from path and name elements
WriteFile()	Function	Writes text to a named file

String Functions

FindWord()	Function	Sets up to find matches to a comma-delimited string
NextWord()	Function	Gets the next word in a comma-delimited string
Len()	Function	Returns the length of a string

Message Statements and System Variables

CreateMessage	Statement	Creates an Outpost message based on parameters
FindMessage()	Function	Searches Outpost for a message
MoveMessage()	Function	Moves an Outpost message to a different folder
NextMessage()	Function	Gets the next Outpost message that matches the search
BBS	System Variable	Holds the BBS name

FROM	System Variable	Holds the FROM address
TO	System Variable	Holds the TO address
SUBJECT	System Variable	Holds the Subject of the message
MESSAGE	System Variable	Holds the body of the message
MTYPE	System Variable	Holds the message type
URGENT	System Variable	Holds the state of the outgoing message Urgent flag.
BBSMSGNO	System Variable	Holds the BBS message number for received messages
RECEIPTS	System Variable	Holds the Receipt flags for outgoing messages
LMI	System Variable	Holds the Local Msg ID for received messages
DATETIME	System Variable	Holds the Date Time string for received messages

Send/Receive Statements and System Variables

SendOnly	Statement	Initiates an Outpost Send Only Session
SendReceive	Statement	Initiates an Outpost Send/Receive Session
Expire()	Function	Sets up a bulletin for deletion from the BBS
BBS	System Variable	Holds the BBS name
TNC	System Variable	Holds the TNC name
MYCALL	System Variable	Holds the Station Identifier (Call Sign)
TACCALL	System Variable	Holds the Tactical Call
RETRIEVE	System Variable	Holds what message types are to be retrieved
FILTER	System Variable	Holds the categories for a Filter Retrieve
SRNOTE	System Variable	Holds the results of a Send/Receive Session

Other System Variables

TRUE	System Variable	Value against which conditions can be checked
FALSE	System Variable	Value against which conditions can be checked
ON	System Variable	Value that can be used to set items
OFF	System Variable	Value that can be used to set items
CRLF	System Variable	Value that causes a carriage return/Line feed on output

5.2 Special Characters

‘ (single quote)	***Description*** The single quote starts the beginning of a comment. Everything after the single quote is part of the comment up until the end of the line. ***Syntax*** ‘ <comment> ***Example*** ‘Comments can begin as the first character x = x+ 1 ‘ or after a statement ***Notes*** 1. All comments are preceded with a single quotation mark.
+ - / *	***Description*** Arithmetic operators: OSL supports the standard arithmetic operations. All precedence rules apply. ***Syntax*** <var> = [var | number] <operand> [var | number] ***Example*** x = x + 1 x = x * (5 - y) / 4 ***Notes*** 1. When an expression has a mix of operators, the precedence of execution is multiple

	and division first, then addition and subtraction. 2. Expressions in parenthesis are always executed first. 3. Space are optional when formatting an expression.

; (semicolon)	***Description***
	Command line continuation. Placing a semicolon at the end of a line allows you to add another command to the same line.
	
	Example
	`x=x+1; y = y – 1`
	`BBS = "K6FB-2"; TNC = "GARAGE-TNC"`
	
	Notes
	1. Care should be taken with this feature since it may contribute to readability problems and debugging your script.

< > =	***Description***
	Relationship operators. These are operators are used as part of conditional tests made with `WHILE…ENDWHILE` and `IF…THEN… ELSE` statements.
	
	Example
	`#1 IF X > 5 THEN`
	`#2 WHILE Y <= 12`
	
	Notes
	1. The following relationship operators are defined:
	`<` less than
	`<=` less than or equal to
	`>` greater than
	`>=` greater than or equal to
	`=` equal to

5.3 Command Reference

Assignments	***Description***
	Assignments are statements that assign a value, variable, result of a function call, or arithmetic operation to another variable.
	
	Syntax
	`<var> = [ number | string | <var> | expression | function ]`
	
	Examples
	`#1. Temp = "Weather.txt"`
	`#2. Result = X / (Y+5)`
	`#3. Fname = NEXTFILE(0)`
	
	Notes
	1. The rules of operational precedence apply to all arithmetic calculations.
	2. There is limited type checking; use caution when mixing strings and numbers in an arithmetic expression.

BBS	***Type***
	System Variable
	
	Description
	Holds the Friendly Name of the BBS. This variable is used by the `CREATEMESSAGE` and `SENDRECEIVE` statements
	
	Syntax
	`BBS = <bbs_name>` Default = blank
	
	Example
	`BBS = "LCARC Path1" ' LCARC's K6FB-2 BBS via AA6WK-7`
	`Print("Checking BBS " & BBS)`
	`SendReceive`
	
	Notes
	1. The value that you assign to the BBS variable is the Friendly name of a BBS that is already defined in Outpost. Connect Names can also be used, but in the event of

	multiple BBS Friendly Name entries with the same Connect Name, the 1st BBS entry will be used (#817, #858, 16-May-10). 2. If this BBS is not set up in Outpost, at the time the Send/Receive session is attempted, Outpost will generate the message: "Either the Station ID, BBS, or TNC is not selected…". This message may not pop to the front; you may need to minimize the Script window to see it.

BBSMSGNO	***Type***
	System Variable
	
	Description
	Holds the BBS message number that was associated with the message retrieved from the BBS.
	
	Example
	`Expire(BBSMSGNO)`
	`SendReceive`
	
	Notes
	1. This field is for display purposes only after retrieving a message.

Beep	***Description***
	Causes the PC to Beep
	
	Syntax
	`Beep`
	
	Example
	`IF x > 5 THEN`
	`  BEEP`
	`ENDIF`
	
	Notes
	1. Also, see the PLAY statement as an alternate audible annunciation option.

Begin	***Description***
	Defines the beginning of the OSL Script statements.
	
	Syntax
	`BEGIN`
	
	Example
	`SCRIPT`
	`VAR x AS NUMBER`
	`BEGIN`
	`  x = 5`
	`  Print("The value of ''x'' is " & x)`
	`END`
	
	Notes
	1. This statement acts as a boundary between all variable declarations and the first script statement.
	2. After pressing NEW, this statement is 1 of 3 statements that are automatically inserted in the new script editing window.
	3. Also, see the SCRIPT, END statements.

Clear	***Description***
	Clears the runtime monitor display. Used primarily for display formatting
	
	Syntax
	`Clear`
	
	Example
	`Loop`
	`  :`
	`  Print("polling BBS " & BBS)`
	`  SendReceive`
	`  Pause(5)`
	`  Clear`
	`Endloop`

	Notes

CreateMessage	***Description***
	Creates a message based on the settings of the message-reserved variables, and writes the message to the Outpost message database.
	
	Syntax
	`CreateMessage`
	
	Example
	`BBS = "K6FB-2"`
	`FROM= "KN6PE"`
	`TO= "K6TEN"`
	`SUBJECT= "Repeater Update"`
	`MESSAGE = ReadFile(RepeaterMessage)`
	`MTYPE = "PRIVATE"`
	`CreateMessage`
	
	Notes
	1. All message reserved variables must be set prior to executing this statement.
	2. Message-reserved variables are: `BBS, FROM, TO, SUBJECT, MESSAGE, MTYPE`
	3. A valid message is written to the Outpost message database and is set for the next send/receive session.
	4. Also, see: `BBS, TNC, MYCALL, TACCALL, RETRIEVE, FILTER`

CRLF	***Type*** System Predefined Variable ***Description*** Contains the 2 characters for Carriage Return and Line Feed. It is used to insert a Carriage Return / Line Feed (same as pressing the Enter Key) in a string so that a single string can display multiple lines. ***Example*** `Msg = "Hi Cap," & CRLF & "Hope all is well." & CRLF & "73, Jim"` ***Notes*** 1. The CRLF is a variable and not part of the string that you define. It is appended to other portions of the string with an "&".
DATETIME	***Type*** System Variable ***Description*** Holds the retrieved message Date time as listed on the BBS and the Outpost message listing. ***Example*** `Print(DATETIME)` ***Notes*** 1. This field is for display purposes after retrieving a message. There is no effect to set this field.

Delete()	***Description***
	Deletes the named file.
	
	Syntax
	`DELETE( file_name )`
	
	Return
	`none`
	
	Example
	#1. `Delete("weather-report.txt")`
	#2. `FName = "weather-report.txt"`
	`Delete(FName)`

	Notes 1. In the event the file does not exist, is open, or is write-protected, the file will not be deleted and an error message will be displayed on the Runtime Monitor.

End	***Description***
	The last statement in the script, Required.
	
	Syntax
	`END`
	
	Example
	`SCRIPT`
	`BEGIN`
	`  Print("Hello World!")`
	`END`
	
	Notes
	1. This statement must be the last statement in the script.
	2. After pressing "NEW", this statement is 1 of 3 statements that are automatically inserted in the new script editing window.
	3. Also, see the `SCRIPT`, `BEGIN` statements.

Exists()	***Description***
	Tests whether the named file exists.
	
	Syntax
	`EXISTS( file_name )`
	
	Return
	`Number: 0 - FALSE`
	`        1 – TRUE`
	
	Example
	`#1. If EXISTS("weather-report.txt") = TRUE then`
	
	`#2. FName = "weather-report.txt"`
	`   Result = Exists(FName)`
	`   IF Result = FALSE THEN`
	`     :`
	
	Notes
	1. The function will return either a 0 or 1 depending on the outcome.
	2. When using with the `IF` statement (1st example), use the System Variables `TRUE` or `FALSE` for the test.

Exit	***Description***
	Terminate Opscripts.exe when encountered (#748)
	
	Syntax
	`EXIT`
	
	Example
	`SCRIPT`
	`BEGIN`
	`  Print("Hello World!")`
	`EXIT                              ' terminate Opscripts`
	`END`
	
	Notes
	1. This command is used whenever you want to run a script from Outpost and terminate scripting when done.
	2. Save your work before running with this command. It will exit without prompting to save your work.

Expire()	***Description*** Delete a bulletin message that belongs to you. ***Syntax*** `EXPIRE( 0 | Bbs_Msg_ID )`

Example

```
FindMessage(1, 4, "*WX ADVISORY*")     ' 1=Intray, 4=Subj Field of Bull name
MsgID = NextMessage(0)

WHILE MsgID > 0                        ' One exists if greater than 0
  IF FROM = "KN6PE" then               ' is it from me? If so, its my Bulletin
    Print("Deleting " & subject)
    EXPIRE(0)                          ' set it up to delete next S/R cycle
    movemessage(MsgID,4)               ' move the message to Archive Folder
  ENDIF

  MsgID = NextMessage(0)               ' get the next match, if any
ENDWHILE
```

Notes

1. Use a "0" with the Expire command to use the BBS message number associated with the last message loaded by the NextMessage function.
2. In the above example, suppose you periodically post a bulletin message that contains the subject line phrase "WX ADVISORY". This lets you find the message again so we can delete it when a new update comes along.
3. The user needs to test to determine if the message being retrieved is in fact a bulletin that (i) exists, and (ii) the user originally posted. Only the bulletin owner can delete a posted bulletin.

FILTER

Description

System Predefined Variable. Holds the string of concatenated filter values that will be used during a Filter Retrieval.

Syntax

```
FILTER = "filter1:filter2:...:filtern"
```

Default = blank

Example

#1. RETRIEVE = "PF"

```
  FILTER = "QST"
```

#2. `FILTER = "LINUX:KEPS:SOCTY"`

Notes

1. `FILTER` must be set if the "F" Filter `Retrieve` option is set.
2. All filters must be separated with colons `":"`.
3. The entire `Filter` assignment enclosed in quotations.
4. Any number of filters can be assigned to the `FILTER` variable.

FindFile()

Description

Searches for and collects all file names that match a particular string pattern.

Syntax

```
FINDFILE( pattern )
```

`pattern`: some or all of the file name to match; use "*" to fill. For instance

`c:\data\WX*.txt` : finds files that start with WX and end with `.TXT`

`*.*` : finds all files in the current directory

Example

```
SCRIPT
VAR NameOnly as string
VAR FullName as string
VAR ctr as number

BEGIN
  ctr = 0
  FINDFILE("c:\data\*.txt")
  FullName = NextFile(0)

  While Exists(FullName) = TRUE
    NameOnly = GetFileName(FullName)

    Print(FullName & " -- " & NameOnly)
```

```
    FullName = NextFile(0)
    ctr = ctr + 1
  ENDWHILE

Print("Files Found: " & ctr)
END
```

Notes

1. This function initializes the File Mask function allowing the **NextFile** function to retrieve each file that matches the mask.
2. Each file returned will contain the equivalent amount of the path as was set up. For instance:

If `FindFile` contains…	then `NextFiles` will include
`scripts\test*.txt`	`script\full_file_name`
`c:\data\*.*`	`c:\data\full_file_name`

3. On entering another file mask, the retrieval is reset.
4. Use the "*" to match any character(s) between characters
5. See the `NextFile` Function

FindMessage()

Description

Searches all Outpost messages that match a particular string pattern.

Syntax

`FINDMESSAGE( <folder>, <field>, <pattern> )`

`folder`: A number corresponding to an Outpost folder to search.
Valid numbers are:

1. InTray
2. Out Tray
3. Sent Folder
4. Archive Folder
5. Draft Folder
6. Deleted Folder
11. Special Folder #1
12. Special Folder #2
13. Special Folder #3
14. Special Folder #4
15. Special Folder #5

`field`: A number corresponding to an Outpost Message field to search.
Valid numbers are:

1. BBS
2. FROM
3. TO
4. SUBJECT
5. MESSAGE

`pattern`: The string pattern to match. Wildcard use (`KN6*`) is allowed.

Characters in Pattern	Matches in String
?	Any single character
*	Zero or more characters
[charlist]	Any single character in charlist
[!charlist]	Any single character not in charlist

Return

none

Example

```
SCRIPT
VAR MsgID as number
VAR ctr as number

BEGIN
  ctr = 0
  FindMessage(1,4,"NOAA*")
```

```
  MsgID = NextMessage(0)

  while msgid > 0
    Print("Found Msg: " & SUBJECT)
    MsgID = NextMessage(0)
    ctr = ctr + 1
  endwhile

  Print("Messages found: " & ctr)
END
```

Notes

1. This function initializes the Message Mask function allowing the **NextMessage** function to retrieve each message that matches the mask. For instance:

If `FindMessage` contains…	then `NextMessage` will include
`CUP*`	`CUPertino, CUP043…`
`*`	`<anything>`

2. On entering another message mask, the retrieval is reset.
3. See the `NextMessage` Function
4. The pattern match is not case sensitive, meaning that a mask of "repeater" will match to a string "REPEATER".
5. Pattern Match Examples:

Pattern	Target String	Result
"F"	"F"	Matches
"f"	"F"	Matches
"FFF"	"F"	No match
"a*a"	"aBBBa"	Matches
"[A-Z]"	"F"	Matches
"[!A-Z]"	"F"	No Match
"a#a"	"a2a"	Matches
"a[L-P]#[!c-e]"	"aM5b"	Matches
"B?T*"	"BAT123Kng"	Matches
"B?T*"	"CAT123Kng"	No Match

6. To match the special characters left bracket ([), question mark (?), number sign (#), and asterisk (*), enclose them in brackets.
7. The right bracket (]) cannot be used within a group to match itself, but it can be used outside a group as an individual character.
8. By using a hyphen (–) to separate the lower and upper bounds of the range, charlist can specify a range of characters.
9. To specify multiple ranges for the same character position, put them within the same brackets without delimiters. Example: [A–CX–Z] matches letters A thru C, and X thru Z.
10. A hyphen (–) can appear either at the beginning (after an exclamation point, if any) or at the end of charlist to match itself.

FindWord()

Description

Sets up to return the individual words found within a comma-delimited string.

Syntax

`FINDWORD( <string> )`

`<string>`: string contains individual words that need to be retrieved

Example

```
SCRIPT
VAR ListOfBBS as string
VAR SingleBBS as string
VAR ctr as number

BEGIN
  ctr = 0
  ListOfBBS = "K6FB-1, W6XSC-1, K6TEN, SANDIEGO"
  FINDWORD(ListofBBS)
  SingleBBS = NextWord(0)

  While LEN(SingleBBS) > 0
    Print("Next BBS name is " & SingleBBS)
    SingleBBS = NextWord(0)
    ctr = ctr + 1
  ENDWHILE
```

	`Print("Number of BBSs Found: " & ctr)`
	`END`
	
	Notes
	1. This function initializes the String Search function allowing the **NextWord** function to retrieve each word from the array of comma-delimited words.
	2. On entering another Word search, the retrieval is reset.
	3. The list of strings must be in a set a quotes. Individual words must be separated by commas.
	4. See the `NextWord` Function

FROM	***Description*** System Predefined Variable. Holds the call sign or tactical calls for the message From field. ***Syntax*** `FROM = "<call_sign>"` Default = blank ***Example*** `From = "KN6PE"` ***Notes*** 1. The FROM assignment is enclosed in quotations.

GetFileName()	***Description***
	Returns the file name portion of a string that includes the file name and path
	
	Syntax
	`<var> = GetFileName(<full_name>)`
	
	Example
	`SCRIPT`
	`Var FullName as String`
	`Var FileName as String`
	
	`BEGIN`
	`FullName = "c:\data\Weather.txt"`
	`FileName = GetFileName(FullName)` returns "Weather.txt"
	`Print(FullName & "  " & FileName)`
	`END`
	
	Notes
	1. This command is useful if you intend to create messages with the subject name embedded in it

If… Then
[Else]
Endif
	Description
	Conditionally executes a block of statements dependent on the state of the condition.
	
	Syntax
	`IF <condition> THEN`
	`  <statements>`
	`[ ELSE`
	`  <statements> ]`
	`ENDIF`
	
	Example
	`#1. If x > 5 THEN`
	`  x = x + 1`
	`ENDIF`
	
	`#2. If x > 5  THEN`
	`  Print(x)`
	`ELSE`
	`  x = x + 1`
	`ENDIF`
	
	Notes

	1. The ELSE statement is optional and not required 2. See Also: WHILE, LOOP
LEN()	***Description*** Returns the length of a string (number of characters) ***Syntax*** <result> = LEN(<string>) result : integer, indicates the number of characters in the string string : the string to be tested ***Example 1*** **SingleBBS =** "K6FB-1" **WordLen = LEN(SingleBBS)** ***Example 2*** **SingleBBS =** "K6FB-1" While LEN(SingleBBS) > 0 : ***Notes***
LMI	***Description*** System Predefined Variable. Holds the Local Message ID (LMI) if enabled in Outpost for incoming messages. ***Syntax*** LMI = "[blank | <LMI value>" Default = depends on Outpost setting ***Example*** No example ***Notes*** 1. This field is for display purposes after retrieving a message. There is no effect to set this field. 2. See the Outpost Users Guide for a description of LMI.
Loop… EndLoop	***Description*** Continuously loops on a block of statements ***Syntax*** LOOP <statements> ENDLOOP ***Example*** LOOP SendReceive Pause(300) ENDLOOP ***Notes*** 1. The only way to exit this loop is to press the "STOP" button on the Runtime control form. 2. See Also: IF, WHILE
MESSAGE	***Description*** System Predefined Variable. Holds the body of the message. ***Syntax*** MESSAGE = "<message text>" Default = blank ***Example*** #1. Message= "Hi Vince, All is still OK here. 73, Jim" #2. Message= ReadFile("Message.txt") ***Notes*** 1. Use a string assigned to MESSAGE for short messages.

	2. Use the `ReadFile()` function to read in the contents of a file to set the message. See Script example #3.
MoveFile()	***Description*** Moves the named file from one location to another. ***Syntax*** `MOVEFILE( <path\file_name>, <dest_path> )` `path\file_name` : The current path and file name of the file to be moved `dest_path` : The Path only of where the file will be moved. Do not include any trailing back slashes ***Return*** none ***Example*** #1. `MoveFile( "c:\data\wx.txt", "c:\data\sent" )` #2. `MoveFile( InName, "c:\data\sent" )` ***Notes*** 1. If the source file is not found, a runtime error will occur and the script will stop. It is recommended that you check for the presence of the file with the `Exists()` function prior to moving or reading a file.
MoveMessage()	***Description*** Moves a message from one Outpost folder to another. ***Syntax*** `MOVEMESSAGE( <msg_id>, <folder_no> )` `Msg_id`: Outpost message pointer. Usually returned by the NextMessage statement `folder_no`: is defined as: 1. InTray 2. Out Tray 3. Sent Folder 4. Archive Folder 5. Draft Folder 6. Deleted Folder 11. Special Folder #1 12. Special Folder #2 13. Special Folder #3 14. Special Folder #4 15. Special Folder #5 ***Return*** none ***Example*** #1. `MoveMessage(MsgID, 4)` message is moved to the Outpost archive folder #2. `MoveMessage(MsgID, 6)` message is moved to the Outpost deleted folder ***Notes*** 1. The Message ID is an internal Outpost identified not typically used in the normal operation from the Outpost forms. From an OSL perspective, the Message ID typically comes from the `NextMessage` function. 2. Any folder value other than those listed above will cause and error and the script to stop.
MTYPE	***Description*** System Predefined Variable. Holds the message type for a message being created. ***Syntax*** `MTYPE= "PRIVATE" | "NTS" | "BULLETIN"` Default = blank ***Example*** #1. `MTYPE = "Private"`

	#2. MTYPE = "NTS" ***Notes*** 1. Only one message type can be set for each message. If more or set, the last Message Type set will the one applied the next time the CreateMessage statement is executed. 2. If not provided, MTYPE defaults to "PRIVATE"
MYCALL	***Description*** System Predefined Variable. Holds the value of the Call Sign that is used to initialize the interface. This variable is used by the SendReceive statement. ***Syntax*** MYCALL = <call_sign> Default = blank ***Example*** MYCALL = "KN6PE" ***Notes*** 1. If left blank, then Outpost will use the currently defined Call Sign as defined from Outpost's **Setup > Identification** form (#758).

NextFile()	***Description***
	Retrieves the next file name that was previously collected by the FindFile function
	
	Syntax
	<Var_name> = NEXTFILE(0)
	
	Return
	String: Next file name (only) that matches the pattern
	If non-blank, valid file name
	If blank (null string), no file found, or reached the end of the list
	
	Example
	SCRIPT
	VAR NameOnly as string
	VAR FullName as string
	VAR ctr as number
	
	BEGIN
	ctr = 0
	FINDFILE("c:\data***.txt")**
	FullName **= NextFile(0)**
	
	While Exists(FullName) = TRUE
	NameOnly = GetFileName(FullName)
	
	Print(FullName & " -- " & NameOnly)
	FullName **= NextFile(0)**
	ctr = ctr + 1
	ENDWHILE
	
	Print("Files Found: " & ctr)
	END
	
	Notes
	1. This function retrieves the next file previously initialized by the FindFile function. The function returns the file name with whatever path was set up as the FindFile() parameter.
	2. The parameter "0" is required. This is for future use.
	3. Each time this function is called, the next file that matches the mask is returned.
	4. When there are no other matches, a blank string is returned. Use the EXISTS() Function to test whether a valid file name was returned.

NextMessage()	***Description*** Retrieves the next message ID that was previously collected by the FindMessage function. ***Syntax*** <Var_name> = NEXTMESSAGE(0)

	Return
	`Integer:` next file that matches the pattern
	If > 0: a valid Outpost message ID
	If = 0: no message found, or reached the end of the list
	
	Example
	`SCRIPT`
	`VAR MsgID as number`
	`VAR ctr as number`
	
	`BEGIN`
	`  ctr = 0`
	`  FindMessage(1,4,"NOAA*")`
	`  MsgID = NextMessage(0)`
	
	`  while MsgID > 0`
	`    Print("Found Msg: " & SUBJECT)    ‘ only print the subjects`
	`    MsgID = NextMessage(0)`
	`    ctr = ctr + 1`
	`  endwhile`
	
	`  Print("Messages found: " & ctr)`
	`END`
	
	Notes
	1. This function retrieves messages based on the selection criteria set up by the `FindMessage()` function.
	2. The Parameter “0” is required. This is for future use.
	3. Each time this function is called, the next message that matches the mask is returned.
	4. When there are no other matches, a value of 0 is returned. Use an `IF… Then` to test whether there is a valid message returned.

NextWord()	***Description***
	Retrieves either the sequentially next word or a specific word that was previously collected by the `FindWord` function
	
	Syntax
	`<Var_name> = NEXTWORD( <index> )`
	`index:` 0 (zero), returns the next word from the list
	1 .. n, returns the indexed word from the list
	
	Return
	`String:` Next word name that was set up
	If non-blank, valid word name
	If blank (null string), no word found, or reached the end of the list
	
	Example 1
	`SCRIPT`
	`VAR SingleBBS as string`
	`VAR ctr as number`
	
	`BEGIN`
	`  ctr = 0`
	`  FINDWORD(“K6FB-1, W6XSC-1, K6TEN, SANDIEGO”)`
	`  SingleBBS = NextWord(0)`
	
	`  While LEN(SingleBBS) > 0`
	`    Print(“Next BBS name is ” & SingleBBS)`
	`    SingleBBS = NextWord(0)`
	`    ctr = ctr + 1`
	`  ENDWHILE`
	
	`Print(“Number of BBSs Found: ” & ctr)`
	`END`
	
	Example 2
	`SCRIPT`
	`VAR SingleBBS as string`
	`VAR ctr as number`
	`BEGIN`
	`  ctr = 4`

```
  FINDWORD("K6FB-1, W6XSC-1, K6TEN, SANDIEGO")
  SingleBBS = NextWord(ctr)

  While ctr > 0
    Print("Next BBS name is " & SingleBBS)
    ctr = ctr - 1
    SingleBBS = NextWord(ctr)
  ENDWHILE
END
```

Notes

1. This function retrieves the next word previously initialized by the `FindWord` function.
2. If the parameter is 0 (zero), then the next sequential word is returned.
3. If the parameter is > 0, then the word that is indexed by the parameter is returned.
4. A parameter is less than 0 or greater than the count of the number of words will returned a blank string.
5. For sequential (0) calls, each time this function is called, the next word is returned. The original string is not affected.
6. When there are no other matches, a blank string is returned. Use the `LEN( )` Function to test whether a string with any length was returned.

Now()

Description

Returns the date and/or time in a format specified by "user."

Syntax

`NOW( "<blank>" | "<format>" )`

Where <format> is:

Date options

Symbol	*Range*
d	1-31 (Day of month, with no leading zero)
dd	01-31 (Day of month, with a leading zero)
w	1-7 (Day of week, starting with Sunday = 1)
ww	1-53 (Week of year, with no leading zero; Week 1 starts on Jan 1)
m	1-12 (Month of year, with no leading zero, January = 1)
mm	01-12 (Month of year, with a leading zero, January = 01)
mmm	Displays 3-character abbreviated month names
mmmm	Displays full month names
y	1-366 (Day of year) This essentially is the Julian day, a continuous count of days since the beginning of the year.
yy	00-99 (Last two digits of year)
yyyy	100-9999 (Three- or Four-digit year)

Time options

Symbol	*Range*
h	0-23 (1-12 with "AM" or "PM" appended) (Hour of day, with no leading zero)
hh	00-23 (01-12 with "AM" or "PM" appended) (Hour of day, with a leading zero)
n	0-59 (Minute of hour, with no leading zero)
nn	00-59 (Minute of hour, with a leading zero)
m	0-59 (Minute of hour, with no leading zero). Only if preceded by h or hh
mm	00-59 (Minute of hour, with a leading zero). Only if preceded by h or hh
s	0-59 (Second of minute, with no leading zero)
ss	00-59 (Second of minute, with a leading zero)

Example

#1. `Now("")`	12/31/2021 5:24:58 PM
#2. `Now("m/d/yy")`	12/31/21
#3. `Now("mm/dd/yyyy")`	12/31/2021
#4. `Now("dd-mmm-yyyy")`	31-Dec-2021

	#5. Now("dd-mmmm-yyyy, hh:mm") 31-December-2021, 17:25 #6. Now("dd-mmm-yyyy, hh:mm AM/PM") 31-Dec-2021, 5:25 PM ***Notes*** 1. A parameter must always be entered with this function.
ON, OFF	***Description*** System Predefined Variable. CONSTANTS ***Notes*** 1. Can be used as a setting and for checking. ON = 1, OFF = 0

OnError	***Description***
	Sets how Opscripts will handle specific types of errors
	
	Syntax
	`ONERROR [ STOP | PAUSE | CONTINUE ]` Default = STOP
	
	Example
	`ONERROR STOP`
	`:`
	`ONERROR CONTINUE ' don't worry on an error`
	`DELETE(fname)`
	
	Notes
	1. Setting a STOP condition will cause the script to report the error on the Runtime form, and stop execution of the script.
	2. Setting a PAUSE condition will pop up a box telling the user to either press **STOP** to stop processing the script, or **RESUME** to continue
	3. Setting a CONTINUE condition will indicate on the Runtime form that an error occurred, and we are continuing anyway.
	4. ONERROR is used to handle the following situations:
	▪ Divide by zero
	▪ RUN(), RUNW(): Running a program, and the program is not found
	▪ DELETE(): deleting a file, but it does not get deleted (could be read-only, or opened to another program)
	▪ MOVEFILE(): Moving a file, but it the destination directory does not exist
	▪ MOVEFILE(): Moving a file, but it the source file does not exist
	▪ READFILE(): Reading a file, but it the file does not exist
	▪ WRITEFILE(): Creating a file, but it did not happen (could be read-only, or opened to another program)
	▪ FINDMESSAGE(): the Folder number is not between 1 and 6 (In tray thru Deleted folder). If the CONTINUE option is set, the Folder value is overridden to a value of "1" (In Tray), and processing continues.
	▪ FINDMESSAGE(): the Field number is not between 1 and 5 (BBS thru MESSAGE) If the CONTINUE option is set, the Field value is overridden to a value of "1" (BBS), and processing continues.
	5. Once an ONERROR condition is set, all errors after that point will be processed with that setting that until a different ONERROR condition is set.

Pause()	***Description*** Causes the script to pause. ***Syntax*** `PAUSE( seconds )` ***Example*** #1. `pause(60)` Pauses for 60 seconds #2. `pvalue = 60` `Pause(pvalue)` Pauses for 60 seconds #3. `pause(0)` Script stops, waits for user interaction ***Notes*** 2. Any value greater than zero will cause the script to pause for the number of seconds indicated. Once this statement is called, the script pauses and the time remaining will count down and be displayed in the lower right portion of the status bar. 3. A value of "0" will cause the script to pause, and requires the user to press the **Resume**

	button on the Runtime Monitor window. This may be useful when there is something that the user needs to do prior to letting the script proceed.
Play()	***Description*** Causes the script to play the named .wav file. ***Syntax*** `PLAY( wav_file_name )` ***Example*** #1. `Play("tada.wav")` #2. `WavName = "tada.wav"` `Play(WavName)` same, with string variable ***Notes*** 4. The file must be locatable either by a fully qualified path or by the system path statement. 5. In the event the file is not found or there is no sound card on your PC, the PC will sound a"beep."

Print()	***Description***
	Prints a string of text to the Runtime Monitor window.
	
	Syntax
	`PRINT( <text_string> )`
	
	Example
	#1. `Print(15)` prints the number 15
	
	#2. `x = 15` set "x" to 15
	`Print(x)` print "x"; same result as above
	
	#3. `Print("Starting Process")` print a string
	
	#4. `x = x + 1` use "x" as a counter
	`Print("Pass #" & x)` print a string and variable
	
	#5. `FName = "Weather.txt"` assign a file name to Fname
	`Print("The file is " & FName)`
	
	Notes
	1. Print will output a single or concatenated string to the runtime monitor window.
	2. Multiple string components can be added and separated by an ampersand "&" sign.
	3. Content can be a mix of explicit string values and variables.

ReadFile()	***Description***
	Reads the content of the named file and assigns its contents to a string variable.
	
	Syntax
	`<Var_name> = READFILE( file_name )`
	
	Return
	`String: file contents`
	
	Example
	#1. `x = ReadFile("c:\data\wx.txt")`
	#2. `MESSAGE = ReadFile(Fname)`
	
	Notes
	1. In the event the file does not exist, or the path is wrong, a "file not found" message is displayed, and the script continues to run.

RECEIPTS	***Description*** System Predefined Variable. Holds the settings for overriding the Receipt Requests for this message.

	Syntax `RECEIPTS = "[ <blank> | R [ D]] "` Default = blank ***Example*** `RECEIPTS = "RD"` ' Request both a Delivery and Read Receipt ***Notes*** 1. The RECEIPTS assignment is enclosed in quotations.
RETRIEVE	***Description*** System Predefined Variable. Holds the string representation of the types of messages to be retrieved. This variable is used by the SENDRECEIVE statement. ***Syntax*** `RETRIEVE = <"P" "N" "B" "F">` Default = "P" ***Example*** #1. `RETRIEVE = "P"` retrieve only Private messages #2. `RETRIEVE = "PNB"` retrieve all message types #3. `RETRIEVE = "PF"` requires Filters to be set ***Notes*** 1. The coding for `RETRIEVE` is as follows: P = Private messages N = NTS messages B = Bulletins F = Filtered 2. If the "F" Filter and "B" Bulletin options are both set, then only the "F" Filter option will be used and the "B" will be ignored. 3. If the "F" Filter option is set, then the Filter string must also be set. If Filter string is not set, then the "F" Filter option is ignored. 4. `RETRIEVE` must be set prior to the next `SendReceive` statement.
Run()	***Description*** Causes the script to run a program, and does not wait for the program to complete before continuing with the script. ***Syntax*** `RUN( exe_file_name )` ***Example*** #1. `Run("notepad.exe")` #2. `Run(PName)` ***Notes*** 1. The executable file must be locatable either by a fully qualified path or by the system path statement. 2. In the event the program does not exist, a "program not found" message is displayed, and the script continues to run.

Runw()	***Description***
	Causes the script to run a program, and will wait for the program to complete before proceeding with the rest of the script.
	
	Syntax
	`RUNW( exe_file_name )`
	
	Return
	`none`
	
	Example
	#1. `Runw("notepad.exe")`
	#2. `Runw(PName)`
	
	Notes
	1. The executable file must be locatable either by a fully qualified path or by the system path statement.
	2. In the event the program does not exist, a "program not found" message is displayed,

	and the script continues to run.

Script	***Description***
	The first OSL statement that appears in the file.
	
	Syntax
	`SCRIPT`
	`BEGIN`
	`  Print("Hello World!")`
	`END`
	
	Notes
	1. This must be the first OSL command in the script file.
	2. After pressing **NEW**, this is 1 of 3 statements that are automatically inserted in the new script editing window.
	3. Also, see: `BEGIN`, `END`

SRNOTE	***Description***
	System Predefined Variable. Holds any Send/Receive Notification message that may occur from the last Send/Receive Session
	
	Syntax
	`SRNOTE = "[ <blank> | <Notification string>]"` Default = blank
	
	Example
	`IF Len(SRNOTE) > 0 then`
	`  Print("Send/Receive problems, message was " & SRNOTE)`
	`ELSE`
	`  Print("Last Send/Receive session was successful!")`
	`ENDIF`
	
	Notes
	1. This field is for display purposes after retrieving a message. There is no effect to set this field.

SendOnly	***Description***
	Initiates an Outpost send only session based on the settings of the system variables. Messages in the out tray will be sent. No check for incoming messages is made.
	
	Syntax
	`SENDONLY`
	
	Example
	`  FROM = "KN6PE"`
	`  TO = "K6KP"`
	`  SUBJECT = "Will miss tonight's net"`
	`  MESSAGE = "Stuck in traffic; start the net without me" & CRLF & "73, Jim o  KN6PE"`
	`  MTYPE = "PRIVATE"`
	`  CREATEMESSAGE`
	`  SENDONLY`
	
	Notes
	1. All session-specific variables must be set prior to executing this statement.
	2. Related System variables used by the `SendOnly` statement are: `BBS, TNC, MYCALL, TACCALL`
	3. Opscripts does not perform any error checking on the existence of the BBS and TNC names entered on these variables. On a Send Only error, Outpost will report the problem, not Opscripts.
	4. Outpost must be running for this statement to work. An error will occur if Outpost is not running.

SendReceive	***Description***
	Initiates an Outpost send/receive session based on the settings of the system variables.
	
	Syntax
	`SENDRECEIVE`
	
	Example
	`  MYCALL = "KN6PE"`

	BBS = "K6FB-2" TNC = "GARAGE-TNC" RETRIEVE = "PB" SENDRECEIVE ***Notes*** 5. All session-specific variables must be set prior to executing this statement. 6. Related System variables used by the SendReceive statement are: BBS, TNC, MYCALL, TACCALL, RETRIEVE, FILTER 7. Opscripts does not perform any error checking on the existence of the BBS and TNC names entered on these variables. On a Send/Receive error, Outpost will report the problem, not Opscripts. 8. Outpost must be running for this statement to work. An error will occur if Outpost is not running.
SUBJECT	***Description*** System Predefined Variable. Holds the subject for this message. ***Syntax*** SUBJECT = "<subject text>" Default = blank ***Example*** #1. Subject = "Status of the W6TDM Repeater" #2. Subject = ReadFile("WX080608.txt") ***Notes*** 1. Subject Line prefixes will be inserted based on Outpost settings.
TACCALL	***Description*** System Predefined Variable. Holds the value of the tactical call. This variable is used by the SendReceive statement. ***Syntax*** TACCALL = <tac_call> Default = "-" ***Example*** #1. TACCALL = "CUPEOC" sets tactical call to CUPEOC #2. TACCALL = "-" turns off tactical call ***Notes*** 1. TacCall is turned off by setting the variable to "-".
TNC	***Description*** System Predefined Variable. Holds the value of the TNC. This variable is used by the SENDRECEIVE statement. ***Syntax*** TNC = <TNC_name> Default = blank ***Example*** TNC = "GARAGE-TNC" ***Notes*** 1. The value that you assign to the TNC variable is the name of a TNC that is already defined in Outpost. For instance, suppose you have a KPC3 that you define in Outpost and give it a name of "GARAGE-TNC". This assigned name is what you assign to the TNC variable. 2. If this TNC is not set up in Outpost, at the time the Send/Receive session is attempted, Outpost will generate the message: "Either the Station ID, BBS, or TNC is not selected…"
TO	***Description*** System Predefined Variable. Holds the call signs or tactical calls of the users for whom this message is intended. ***Syntax*** TO = "<call_sign> [, 2nd_address]" Default = blank

	Example #1. `To = "KN6PE"` #2. `To = "KN6PE, SMTP:kn6pe@arrl.net"` #3. `DistList = "K6KP, W6TDM, SMTP:kn6pe@arrl.net"` `To = DistList` ***Notes*** 1. All standard address rules are in force when addressing messages to a Winlink station.
TRUE, FALSE	***Description*** System Predefined Variable, CONSTANTS, used as part of a conditional test. ***Example*** `IF Exists(Fname) = TRUE then` ***Notes*** 1. TRUE and FALSE can be used to check for this case. Additional functions may be added in the future to take advantage of this.
URGENT	***Description*** System Predefined Variable. Holds the outgoing message URGENT Flag. ***Syntax*** `URGENT = TRUE | FALSE` Default = FALSE ***Example*** #1. `URGENT = TRUE` #2. `URGENT = FALSE` ***Notes*** 1. Once the URGENT flag is set, it is applied to all subsequent created messages. It is recommended that you explicitly declare whether a message should be URGENT or not. 2. Initially, URGENT defaults to FALSE.

ValidFileName()	***Description***
	Creates a valid full-path file name from a path and name components. This is typically used when creating files from Outpost messages, and there may be invalid file name characters in the Subject name.
	
	Syntax
	`<Var_name> = ValidFileName(<string>)`
	
	Example
	`SCRIPT`
	`Var FullName as String`
	`Var FixedName as String`
	
	`BEGIN`
	`SUBJECT = "CUP103: c:\data\Weather report.txt"`
	`FixedName = ValidFileName(SUBJECT)`
	`FullName = "c:\data\" & FixedName`
	`Print(FullName)`
	`END`
	
	Notes
	1. The following 9 characters work for Outpost subjects but are invalid file name characters: **: / \ * ? | < > "**
	2. The "**:**" character will be replaced with a "**;**"
	3. The **/ \ * ? | < > "** characters will be replaced with a "**~**"
	4. So, in the above example, the `FixedName` is set to…
	`CUP103; c;~data~Weather report.txt`

Var	***Description***
	Declares a user-defined variable that can be subsequently assigned and manipulated
	
	Syntax
	`VAR <var_name> AS [STRING | NUMBER]`
	
	Example
	`Script`

```
VAR Fname as string
VAR Shelter24 as string
VAR x as number
BEGIN
```

Notes

1. All user-defined variables must be defined after the SCRIPT statement and before the BEGIN statement.
2. Variable names must start with a letter and follow with any combinations of letters, numbers and the underscore (_) character. All other punctuation are not allowed.
3. Var types are String or Number

While… Endwhile

Description

Executes a block of statements as long as the condition is true.

Syntax

```
WHILE <condition>
   <statements>
ENDWHILE
```

Example

```
SCRIPT
VAR Fname as string

BEGIN
  FINDFILE("c:\data\" & "*.txt")
  Fname = NextFile(0)

  While Exists(Fname) = TRUE
    Print(Fname)
    Fname = NextFile(0)
  ENDWHILE
END
```

Notes

1. See Also: IF, LOOP

WriteFile()

Description

Writes data to a named file

Syntax

```
WRITEFILE( <data>, <file_name> )
```

data: a text string or variable of the data to be written

file_name: a string or variable of the name of the file to be created

Example

```
SCRIPT
VAR MsgID as number

BEGIN
  FindMessage(1,4,"NOAA*")    ' set up the msg search
  MsgID = NextMessage(0)      ' loads the current msg

  while msgid > 0
    Print("Found Msg: " & SUBJECT)
    WriteFile(MESSAGE, Subject & ".txt")
    MsgID = NextMessage(0)
  endwhile

END
```

***Example** #2*

```
' Append a line of text to an existing file
SCRIPT
  VAR Fname as string       ' Name of a file
  VAR Fdata as string       ' Contents of the file

BEGIN
  Fname = "C:\data\Master.ini" ' set the file name
  Fdata = ReadFile(Fname)      ' Read the file contents
```

```
  Fdata = Fdata & CRLF & "Cmd=0"  ' append a line of text
  WriteFile(Fdata, Fname)      ' Write the new file contents
END
```

Notes

1. Any content can be written to a file. If the file already exists, it will first be deleted.
2. The data to be written can be the explicit string in quotations, or a variable containing the string.
3. In the above example, the `NextMessage` loads the next message and all its variables into the system variables: `BBS, FROM, TO, SUBJECT, MESSAGE`. The `WriteFile` statement writes the content of the variable `MESSAGE` (the current Outpost message) to the file by the name "`<subject>.txt`"; the file has the subject string in the title.
4. In the 2nd example, this is a way to append data to a file. Essentially, read the contents, append the addition, and write it back.