CA Client Automation

Desktop Management Scripting Language

12.9

CA Technologies Product References

This documentation set references to the following CA products:

- CA Advantage® Data Transport® (CA Data Transport)
- CA Asset Intelligence
- CA Asset Portfolio Management (CA APM)
- CA Business Intelligence
- CA Common Services™
- CA Desktop Migration Manager (CA DMM)
- CA Embedded Entitlements Manager (CA EEM)
- CA Mobile Device Management (CA MDM)
- CA Network and Systems Management (CA NSM)
- CA Patch Manager
- CA Process Automation
- CA Service Desk Manager
- CA WorldView™

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Desktop Management Scripting Language

The Scripting Language provides a common means of executing commands on agents. It is a programming tool with the ability of Desktop Management components to run agent commands that, for example, let you perform software maintenance on the asset management agents. When you are familiar with the scripting commands, you can use them on all agent platforms and operating systems.

The core language of the script includes the usual control statements, expressions, and variables. The core language has libraries of functions that enable you to perform general and specific tasks needed by the item procedures.

By using the language features, you can make scripts portable between the different target operating systems.

Identifiers

Identifiers are constants, variables, and functions that you use in your program. An identifier can contain a maximum of 64 characters and must begin with a letter or an underscore. An identifier cannot contain spaces and parentheses. Identifiers are not case-sensitive.

Examples:

MyVar
MyString

Numbers

DMS uses integer numbers of 32 bits. A number can be in the range of -2147483647 to 2147483648.

Comments

A comment is a line that begins with REM or with a single quotation mark ('). You can also insert REM or a single quotation mark in a line, so that the rest of the line is ignored.

Example: Comments

```
REM This line is a comment.

'This line is a comment
Dim x as Integer ' This is a comment
```

Error Codes (Desktop Management Scripting Language)

The DMS language error codes are as follows:

Error Code	Description
1	Syntax error
2	Invalid variable index type
3	Unknown variable type
4	Memory shortage
5	Duplicate variable name
6	Variable name expected
7	Unknown variable
8	Unknown function
9	Variable is required
10	Variable is not initialized
11	AS expected before type name
12	Unknown variable type name
13	Invalid variable name
14	Cannot open the script file
15	Duplicate label name
16	Missing THEN in IF .. THEN block

Error Code	Description
17	Ordinal expression is required
18	Missing END IF in IF .. THEN block
19	Missing END WHILE in WHILE ... END WHILE block
20	Missing UNTIL in REPEAT ... UNTIL block
21	Missing TO in FOR .. NEXT block
22	Missing NEXT in FOR .. NEXT block
23	Syntax error in function definition
24	Missing END FUNCTION in function definition
25	Unknown EXIT statement type
26	Invalid mix of data types in expression
27	Integer or String expression expected
28	Unknown expression operation
29	Data types not compatible
30	Invalid operation for this data type
31	Division by zero
32	Basic types expected
33	Expression expected
34	Variable reference expected
35	Argument mismatch in call to function
36	Invalid redirection
37	Preprocessor variable name expected
38	Syntax error in preprocessor expression
39	Preprocessor syntax error
40	#ELSE without a corresponding #IF found
41	#ELSEIF without a corresponding #IF found
42	#ENDIF without a corresponding #IF found
43	Invalid preprocessor directive
44	Unknown preprocessor directive
45	Invalid function format
46	Unable to load library file

Error Code	Description
47	Unable to locate function
48	Character or String expression expected
49	Immediate value expected
50	Unable to evaluate constant
51	Invalid variable type in argument
52	Unterminated text string constant
53	Character constant expected
54	Un-terminated character constant
55	Expression syntax error
56	User break in script debug session
57	Program terminated in script debug session
58	Invalid GOTO destination
59	Script symbol expected
60	Unable to evaluate variable
68	Cannot load exit list
69	Cannot register class
70	Failed to load DLL
73	Invalid array type
74	Invalid structure member type
75	Array index out of range
76	Missing array size field
77	Warning: Multiple declaration of function "<function name>" detected. The Repetition is ignored.
78	Script file missing
79	Locale problem detected
80	Interpreter canceled by operator
81	Warning: Masked quotes (\") detected in script. Simple quotes (") assumed.
82	Insufficient memory. DMS Interpreter abnormally terminated.
83	Internal error
84	Script consists of more than 64 K lines

Error Code	Description
85	Include File not Found

Declare Variables

Declare a variable before using it in a script and declare a variable before assigning a value to it.

To declare script variables, use the DIM keyword.

The basic types of variables and their default values are as follows:

Boolean

unsigned char (False)

Integer

long (0)

String

char * ("")

Char

wchar_t (0)

Byte

unsigned char (0)

Word

unsigned short (0)

Short

short (0)

Examples:

```
DIM MyIntegerVar as Integer
DIM MyStringVar as String
MyIntegerVar = 1234
MyStringVar = "This is a string."

DIM C1, C2, C3 as Char
C1 = 0x61
C2 = 0x0061
C3 = "a"
rem C1, C2, C3 - all the same character
```

Labels

A label is an identifier that conforms to the following rules:

- Begin or terminate a label by a colon ":".
- Use letters, digits, or underscores only.
- Use in a GOTO statement to transfer program execution from some point in the program to the statement prefixed by the label.

Note: Labels are not case-sensitive.

Example:

```
Dim Number As Integer
Number = 1
IF Number=1 Then GOTO end
PRINT "Number=0"
end:
```

Declare Arrays

An array is an ordered arrangement of data elements. The script interpreter can only process one-dimensional arrays (vectors).

Example:

A vector of 12 elements, where the first element a[0] has the (numeric) value 444.

```
Dim ar[12] as integer
    0 - 11
    a[0] = 444
```

Note: Only basic types can declare arrays, but you can use arrays to declare user types.

Correct Examples

```
DIM a[25] as Integer

DIM b[100] as Boolean
```

Wrong Examples

```
TYPE T1
    A as Integer
    B as Char
END TYPE


DIM C[25] as T1
```

Expressions

Expressions consist of operators and operands.

Operators include: AND OR * / + - = <> < > <=

The rules of precedence that apply to operators follow the rules of algebra. In other words, all operators take two operands.

Note: The NOT operator does not exist, but you can use the predefined function NOT.

If the operands are of type String, the + operator concatenates them:

```
DIM FirstName as String
DIM LastName as String
DIM FullName as String
DIM Space as String
FirstName="Mary"
LastName="Johnson"
Space=" "
FullName=FirstName+Space+LastName
```

The expression FirstName+Space+LastName evaluates to "Mary Johnson" and thus the contents of the string FullName is "Mary Johnson."

If you use the **+** operator with operands of type Integer, the script interpreter adds the operands:

```
DIM X as Integer
DIM Y as Integer
DIM Z as Integer
X=2
Y=3
Z=X+Y
```

The variable **Z** contains the integer value 5.

The result type of an expression is the largest type of operand.

```
Byte+Word = Word
```

```
Char+String = String
```

Special Characters

In some programming languages such as C, you use the backslash ("\)" to mark special characters. Desktop Management Scripting does not support the backslash as an escape sign but only as a normal character to ease path name and file handling. To enter special characters, code the hex value of that character.

For example, to add a newline to a string in a Windows environment, add the following lines to the script:

```
Dim LF, CR As Char
Dim NL, str As String
LF = 0x0A
CR = 0x0D
NL = CR + LF
 .
 .
 .
str = str + CRLF;
```

In the UNIX environment, the CR is obsolete; and replace the NL initialization in the previous example, as follows:

```
NL = LF
```

Frequently used codes are as follows:

```
Tabulator (HT):          0x09
Line Feed (LF):          0x0A
Carriage Return (CR): 0x0D
```

The Desktop Management Scripting provides a constant for newline that is named NEWLINE$. The interpreter takes care of the correct initialization and the value of this constant depending on the environment it is running. You only use the constant in your script. The previous example now looks as follows:

```
Dim str As String
 .
 .
 .
str = str + NEWLINE$
```

Statements

Statements describe the actions you want the script to perform.

More Information:

Conditional Statements (see page 15)

Assignment Statements (see page 15)

Function Call Statements (see page 17)

GOTO Statements (see page 18)

Repetitive Statements (see page 18)

Assignment Statements

The assignment statement is the most basic scripting language statement.

Example:

```
DIM x as Integer
DIM y as Integer
DIM z as Integer
x=1
y=x+z
```

Conditional Statements

The scripting language has one type of conditional statement - the IF..THEN..ELSE statement. The syntax of this statement is as follows:

```
IF <expression> THEN
   <statement block>
[ELSE
      <statement block>]
ENDIF
```

You can use the conditional statement without ENDIF, if the whole statement fits into one line:

```
IF <expression> THEN <statement block>
```

The result of the expression must be of type Integer.

- If the expression is TRUE (nonzero value), the statement following THEN is executed.
- If the expression is FALSE (zero value) and the ELSE part is present, the statement following ELSE is executed.
- If the ELSE part is not present, program execution continues at ENDIF.

Note: Instead of ENDIF you can use End If.

Another form of the conditional statement deals with nested conditional statements. Instead of coding nested conditional statements as shown in the first code sample that follows, you can code them as shown in the second sample.

```
DIM os As String
.
.
.
If os = "AIX" Then
        aixRun()
Else
        if os = "HPUX" Then
                rc = hpRun()
        Else
                if os = "SOLARIS(sparc)" Then
                                solarisRun()
                else
                                print("unknown os type: " + os)
                End If
        End If
End If
.
.
.
```

Second sample:

```
DIM os As String
.
.
.
If os = "AIX" Then
        aixRun()
Elseif os = "HPUX" Then
        hpRun()
Elseif os = "SOLARIS(sparc)" Then
        solarisRun()
else
        print("unknown os type: " + os)
End If
.
.
.
```

Example:

```
DIM ProgName as String
DIM ProgPath as String
ProgPath="C:\AVIRUS"
IF InStr(ossystem,"DOS") THEN
ProgName=ProgPath + "FINDVIRU.EXE"
ELSE
ProgName=ProgPath + "WFINDVIR.EXE"
ENDIF
end:
```

Function Call Statements

You can use a function **call** as a statement where the function result is ignored.

Example:

```
Print("message: Starting section 2")
```

GOTO Statements

You can use the GOTO statement to transfer the program execution from one point in a program to another. The name of the label where the processing is to continue must follow the statement.

You cannot use the GOTO statement to jump into a loop (FOR, WHILE, REPEAT), function block, or IF (ELSE) block from outside.

Example:

```
GOTO MyLabel
```

Repetitive Statements

Repetitive statements repeat actions while or until a condition is true.

FOR..NEXT..STEP Statements

The syntax of this type of statement is as follows:

FOR *variable=expression* TO *expression*
  [STEP *expression*]
    statement block
NEXT *variable*

Where, *expression* must evaluate into an integer.

Note: You can use EXIT FOR in a FOR..NEXT..STEP statement to exit the FOR loop.

Example:

The statement prints each of the numbers 2, 4, 6, and 8 on a separate line and then prints all four numbers on one line.

```
DIM A as String
DIM J as Integer
A=""
FOR J=2 TO 8 STEP 2
A=A+Str(J)
PRINT Str(J)
NEXT J
PRINT A
end:
```

WHILE..WEND Statements

A WHILE..WEND statement contains an expression that controls the repetitive execution of a statement block.

The expression result must be of type Integer. The statement block is run repeatedly as long as the expression evaluates to true (nonzero). If the expression is false (zero) initially, the statement block is not run.

The statement syntax is as follows:

```
WHILE <expression>
<statement block>
WEND
```

EXIT WHILE exits the WHILE..WEND loop.

Example:

```
DIM FileName as String
DIM NoOfLines as Integer
DIM TmpLine as string
DIM hFile as Integer
FileName="C:\AUTOEXEC.BAT"
NoOfLines=0
hFile=OpenFile(FileName,0,0)
WHILE NOT(Eof(hFile))
  ReadFile(hFile,TmpLine)
  NoOfLines=NoOfLines+1
WEND
CloseFile(hFile)
NoOfLines=NoOfLines-1
PRINT FileName + " Contains "+ str(NoOfLines) + " lines."
end:
```

REPEAT..UNTIL Statements

A REPEAT..UNTIL statement contains a statement block. An expression controls repetitive runs by an expression.

The expression result must be of type Integer. The statement block runs repeatedly until the expression evaluates to true (nonzero).

This statement syntax is as follows:

```
REPEAT
statement block
UNTIL expression
```

Example: REPEAT ... UNTIL statements

```
DIM FileName as String
DIM NoOfLines as Integer
DIM TmpLine as string
DIM hFile as Integer
FileName="C:\AUTOEXEC.BAT"
NoOfLines=0
hFile=OpenFile(FileName,0,0)
REPEAT
 ReadFile(hFile,TmpLine)
 NoOfLines=NoOfLines+1
UNTIL Eof(hFile)
CloseFile(hFile)
NoOfLines=NoOfLines-1
PRINT FileName + " Contains "+ str(NoOfLines) + " lines."
end:
```

Functions

The Desktop Management Scripting language functions are grouped into related types.

Note: The scripting language is not case-sensitive. The contents of character strings are case-sensitive (for example, the strings "ABCD", "Abcd", and "abcd" are different).

Common Configuration Functions

You can run the common configuration functions on Windows and Unix platforms.

Note: These functions are new in CA Client Automation and do not work with older versions of the Script Interpreter.

CcnfGetParameterInt - Read Common Configuration Parameter Integer Value

Valid on Windows and UNIX

The CcnfGetParameterInt function reads the common configuration parameter integer value.

Function format:

```
CcnfGetParameterInt(Parametername as string, Value as integer) as Boolean
```

Parametername

Defines the name of a common configuration parameter including its absolute path (for example: "itrm/rc/protocols/tcp/connectiontimeout").

Value

Defines the integer value of Parametername.

The return value of the function is Boolean. If the function is successful, it returns TRUE and value contains the value of Parametername. If the function is not successful, an error is reported in the log file.

Example:

This example uses a comstore parameter:

```
Dim pNum as integer
if CcnfGetParameterInt("itrm/usd/shared/localtimedeviation ", pNum) then
Print("itrm/usd/shared/localtimedeviation = '"" + val(pNum) + '"" )
else
Print("CcnfGetParameterInt failed")
endif
```

CcnfGetParameterStr - Read Common Configuration Parameter String Value

Valid on Windows and Unix

The CcnfGetParameterStr function reads the common configuration parameter string value.

Function format:

```
CcnfGetParameterStr(Parametername as string, Value as string) as boolean
```

Parametername

Name of a Common Configuration Parameter including its absolute path (for example, "itrm/rc/protocols/encrypt/dll")

Value

String value of Parametername.

The return value of the function is boolean. If the function is successful, it returns TRUE and Value contains the value of Parametername. If the function is not successful, an error is reported in the log file.

Example:

This example uses a comstore parameter:

```
Dim pText as string
if CcnfGetParameterStr("itrm/usd/shared/nos", pText) then
Print("itrm/usd/shared/nos = """ + pText + """" )
else
Print("CcnfGetParameterStr failed")
endif
```

CcnfSetParameterInt - Modify the Value of an Existing Common Configuration Parameter (type string)

Valid on Windows and UNIX

The CcnfSetParameterInt function modifies the value of an existing integer type common configuration parameter.

This common configuration parameter function has the format:

```
CcnfSetParameterInt(Parametername as string, Value as integer) as boolean
```

Parametername

Name of a Common Configuration Parameter including its absolute path, for example: "itrm/rc/protocols/tcp/connectiontimeout"

Value

String value of Parametername

The return value of the function is a boolean. If successful, the function returns TRUE. If the function is not successful, an error is reported in the log file.

The correctness of the value is not verified.

Example:

This example uses a comstore parameter:

```
if CcnfSetParameterInt("itrm/usd/shared/localtimedeviation ", 9) then
Print("value of itrm/usd/shared/localtimedeviation was modified" )
else
Print("CcnfSetParameterInt failed")
Endif
```

CcnfSetParameterStr - Modify the value of an existing Common Configuration Parameter (type string)

Valid on Windows and Unix

The CcnfSetParameterStr function modifies the value of an existing common configuration parameter of type string.

This common configuration parameter function has the format:

```
CcnfSetParameterStr(Parametername as string, Value as string) as boolean
```

Parametername

Name of a Common Configuration Parameter including its absolute path (for example, "itrm/rc/protocols/encrypt/dll")

Value

String value of Parametername

The return value of the function is a boolean. If successful, the function returns TRUE. If the function is not successful, an error is reported in the log file.

The correctness of the value is not verified.

Example: CcnfSetParameterStr function

This example uses a comstore parameter that is described in the chapter Mapping Between asm.cnf and comstore.xml in the *Implementation Guide*.

```
if CcnfSetParameterStr("itrm/usd/shared/nos", "newValue") then
Print("value of itrm/usd/shared/nos was modified" )
else
Print("CcnfSetParameterStr failed")
endif
```

Intellisigs Functions

When using custom Intellisigs, call the DMScript functions within your Intellisig script for reporting software records that are detected on the agent computer. DMScript provides built-in functions that write detected software records to an output file.

The Intellisig script must call the functions in the following order:

1. OpenDetectedSoftwareOutputFiles
2. CreateDetectedSoftwareProduct
3. CreateDetectedSoftwareRelease
4. CreateDetectedSoftwarePatch
5. CreateDetectedSoftwareReleaseInstance
6. CloseDetectedSoftwareOutputFiles

OpenDetectedSoftwareOutputFiles

The OpenDetectedSoftwareOutputFiles function creates an empty software detection output file for the Intellisig. A software detection output file contains the records detected for an Intellisig. When the Intellisig script executes at the agent computer, the OpenDetectedSoftwareOutputFiles function creates an output file to store the software detected by the Intellisig. For example, if an Intellisig includes definitions for all Adobe products, releases, and patches, you can write all the Adobe detection records to the same output file.

Note: The Intellisig script must call the OpenDetectedSoftwareOutputFiles function regardless of whether the Intellisig detects any software or not. An Intellisig that does not create an output file generates an error during execution.

Function format:
OpenDetectedSoftwareOutputFiles (IntellisigUUID as String, Version as String, Name as String)as integer

Example:
OpenDetectedSoftwareOutputFiles ("A7C1E14A-7C93-4E17-B4E5-45B796717F49", "V1", "OS Detection for Windows")

Input Parameters

This function has the following input parameters:

IntellisigUUID

Specifies the Universal Unique Identifier (UUID) of the Intellisig. The function creates a software detection output file with the given UUID as the file name.

IntellisigVersion

Specifies the version number of the Intellisig.

IntellisigName

Specifies the name of the Intellisig.

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_BADARGS

Indicates that one or more mandatory parameters passed to the function are blank.

CASWDETECT_FILEERROR

Indicates that the function was unable to open the file. See the log file for more information.

CreateDetectedSoftwareProduct

The CreateDetectedSoftwareProduct function adds a record to the software detection output file when the script detects a product that is related to the Intellisig.

Function format:

CreateDetectedSoftwareProduct (ProductName as String, ProductVersion as String, OptionalProperties as String) as an integer

Example:

```
CreateDetectedSoftwareProduct(("Microsoft Windows 7 Ultimate", "6.1",
"VersionNumber=6.1  |Manufacturer=Microsoft Corporation|Category=Operating Systems
|Description=The Microsoft Windows 7 Product")
```

Note: Call this function before calling the CreateDetectedSoftwareRelease function, which creates the release of the product.

Input Parameters

This function has the following input parameters:

ProductName

Specifies the product name of the detected software.

Note: The product name must be unique within the Intellisig. Two products with the same name within an Intellisig are treated as the same product regardless of optional parameters.

ProductVersionLabel

Specifies the version label of the product.

Note: The version label is used to identify the product together with its name. An empty value is allowed and is treated as a product with an empty version.

OptionalProperties

Specifies the optional properties that are associated with the detected software product. Following optional properties are available for products:

VersionNumber

Specifies the version number of the product.

Language

Specifies the product language.

Bitness

Specifies whether the product uses 32-bit or 64-bit architecture.

Architecture

Specifies the architecture name that the product uses.

Manufacturer

Specifies the name of the product manufacturer.

Category

Specifies the name of the product category.

ManufacturerUUID

Specifies the UUID of the product manufacturer.

CategoryUUID

Specifies the UUID of the product category.

Description

Specifies the description of the product.

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_BADARGS

Indicates that one or more mandatory parameters passed to the function are blank.

CASWDETECT_FILE_ERROR

Indicates that the function was unable to write to the file. See the log file for more information.

CreateDetectedSoftwareRelease

The CreateDetectedSoftwareRelease function adds a record to the software detection output file when the script detects a release of a product on the agent computer.

Note: DMScript adds the record only if it finds the related product record in the MDB.

This function has the following format:

CreateDetectedSoftwareRelease (ProductName as String, ProductVersionLabel, as String, ReleaseName as String, ReleaseVersionLabel as String, OptionalProperties as String) as an integer

Example:

CreateDetectedSoftwareRelease ("Microsoft Windows 7 Ultimate", "6.1", "Microsoft Windows 7 Ultimate x64 64 en-us", "6.1.7600", "VersionNumber=6.1.7600 |Language=en-us |Bitness=64 |Architecture=x64 |Manufacturer=Microsoft Corporation|Category=Operating Systems |Description=The Microsoft Windows 7 Release")

Input Parameters

This function has the following input parameters:

ProductName

Specifies the product name that the release belongs to. You must have called the CreateDetectedSoftwareProduct function for this product within the script.

ProductVersionLabel

Specifies the version label of the product.

Note: The version label is used to identify the product together with its name. An empty value is allowed and is treated as a product with an empty version.

ReleaseName

Specifies the name of the discovered software release.

Note: A release can have the same name as the product. However, Intellisigs support different release names to help associate multiple releases with the same product. The release name and version label must be unique within each Intellisig chain. Two releases with the same name within an Intellisig are treated as the same release only if they share the same parent definitions, regardless of optional parameters. If the same release is detected using two different Intellisigs, two separate detected records are created, one for each Intellisig.

ReleaseVersionLabel

Specifies the version label of the release.

Note: The version label is used to identify the release together with its name. An empty value is allowed and is treated as a release with an empty version.

OptionalProperties

Specifies the optional properties associated with the discovered software release. Following optional properties are available for releases:

VersionNumber

Specifies the version number of the release.

Language

Specifies the language in which the release is installed.

Bitness

Specifies whether the release uses 32-bit or 64-bit architecture.

Architecture

Specifies the architecture name.

Manufacturer

Specifies the name of the release manufacturer.

Category

Specifies the release category.

ManufacturerUUID

Specifies the UUID of the release manufacturer.

CategoryUUID

Specifies the UUID of the release category.

Description

Specifies the description of the release.

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_BADARGS

Indicates that one or more mandatory parameters passed to the function are blank.

CASWDETECT_FILE_ERROR

Indicates that the function was unable to write to the file. See the log file for more information.

CreateDetectedSoftwarePatch

The CreateDetectedSoftwarePatch function adds a record to the software detection output file when the script detects a patch of a particular release. Add this function for every patch that you have included in the Intellisig script.

Function format:

CreateDetectedSoftwarePatch (ProductName as String, ProductVersionLabel, as String, ReleaseName as String, ReleaseVersionLabel as String, PatchName as String, PatchVersionLabel as String, OptionalProperties as String) as an integer

Example:

CreateDetectedSoftwarePatch ("Microsoft Windows 7 Ultimate", "6.1", "Microsoft Windows 7 Ultimate x64 64 en-us", "6.1.7600", "KB971033 x64 64 en-us", "Language=en-us |Bitness=64 |Architecture=x64 |Manufacturer=Microsoft Corporation|Category=Operating Systems |Description=The Microsoft Windows 7 Activation Checker Update")

Note: Called this function after calling the CreateDetectedSoftwareRelease function within each script.

Input Parameters

This function has the following input parameters:

ProductName

Specifies the name of the product that the patch belongs to. You must have called the CreateDetectedSoftwareProduct function for this product within the script.

ProductVersionLabel

Specifies the version label of the product.

Note: The version label is used to identify the product together with its name. An empty value is allowed and is treated as a product with an empty version.

ReleaseName

Specifies the name of the release that the patch belongs to. You must have called the CreateDetectedSoftwareRelease function for this release within the script.

ReleaseVersionLabel

Specifies the version label of the release.

Note: The version label is used to identify the release together with its name. An empty value is allowed and is treated as a release with an empty version.

PatchName

Specifies the name of the detected software patch.

Note: A patch cannot have the same name as the release or product to which it belongs. The patch name and version label must be unique within each Intellisig software definition chain. Two patches with the same name within an Intellisig are treated as the same patch only if they have the same parent definitions, regardless of optional parameters. If the same patch is detected using two different Intellisigs, two separate detected records are created, one for each Intellisig.

PatchVersionLabel

Specifies the version label of the patch.

Note: The version label is used to identify the patch together with its name. An empty value is allowed and is treated as a patch with an empty version.

OptionalProperties

Specifies the optional properties associated with the detected software patch. Following optional properties are available for patches:

VersionNumber

Specifies the version number of the patch.

Language

Specifies the language in which the patch is installed.

Bitness

Specifies whether the patch uses 32-bit or 64-bit architecture.

Architecture

Specifies the architecture name that the patch uses.

Manufacturer

Specifies the patch manufacturer name.

Category

Specifies the patch category.

ManufacturerUUID

Specifies the UUID of the patch manufacturer.

CategoryUUID

Specifies the UUID of the patch category.

Description

Specifies the description of the patch.

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_BADARGS

Indicates that one or more mandatory parameters passed to the function are blank.

CASWDETECT_FILE_ERROR

Indicates that the function was unable to write to the file. See the log file for more information.

CreateDetectedSoftwareReleaseInstance

The CreateDetectedSoftwareReleaseInstance function adds a record to the software detection output file when the script detects an instance of a particular release.

Function format:

CreateDetectedSoftwareReleaseInstance (ProductName as String, ProductVersionLabel as String, ReleaseName as String, ReleaseVersionLabel as String, OptionalProperties as String) as an integer.

Example:

CreateDetectedSoftwareReleaseInstance ("Microsoft Windows 7 Ultimate", "6.1", "Microsoft Windows 7 Ultimate x64 64 en-us", "6.1.7600", "Origin=Forward Inc | TrustLevel=5 | InstallPath=C:\Windows | SerialNumber=1234-567-890414-86668 | LastAccessed=2011-11-29:-12:30 |")

Note: Call this function after calling the CreateDetectedSoftwareRelease function within each script. The CreateDetectedSoftwareReleaseInstance function can be called as many times as there are instances and each instance gets a separate discovered software record.

Note: If two different Intellisigs detect the same software instance, two discovered software records are created.

Input Parameters

This function has the following input parameters:

ProductName

Specifies the product name that the release belongs to. You must have called the CreateDetectedSoftwareProduct function for this product within the script.

ProductVersionLabel

Specifies the version label of the product.

ReleaseName

Specifies the name of the discovered software release.

ReleaseVersionLabel

Specifies the version label of the release.

OptionalProperties

Specifies the optional properties associated with the discovered software instance. The following optional properties are available for instances:

ProductGUID

Specifies the product UUID as detected by the script. For example, the product GUID of an MSI package.

Label

Specifies a unique label for this instance. For example, the Microsoft SQL Server instance name.

Origin

Specifies the name of the Intellisig creator.

TrustLevel

Specifies the trust level of the creator of the Intellisig, script, or both.

InstallPath

Specifies the path to the product installation directory or the executable on the agent computer.

SerialNumber

Specifies the serial number of the instance.

LastAccessed

Specifies date and time when the instance was last accessed. Specify the value in the following format: yyyy-mm-dd-hr:mm. This value is assumed to be in local time.
Note: All other functions in dmscript that deal with time, operate in local time. Dmscript converts the time value to Unix format (seconds since 1-1-1970 UTC) before output. DSM Explorer displays this using the local time zone on the machine on which it is running.

CustomData

Specifies any other custom data you want store for the instance.

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_BADARGS

Indicates that one or more mandatory parameters passed to the function are blank.

CASWDETECT_FILE_ERROR

Indicates that the function was unable to write to the file. See the log file for more information.

CreateDetectedSoftwarePatchInstance

The CreateDetectedSoftwarePatchInstance function adds a record to the software detection output file when the script detects an instance of a particular patch.

Function format:

CreateDetectedSoftwarePatchInstance (ProductName as String, ProductVersionLabel, as String, ReleaseName as String, ReleaseVersionLabel as String, PatchName as String, PatchVersionLabel as String, OptionalProperties as String) as an integer

Example: CreateDetectedSoftwarePatchInstance

CreateDetectedSoftwarePatchInstance ("Microsoft Windows 7 Ultimate", "6.1", "Microsoft Windows 7 Ultimate x64 64 en-us", "6.1.7600", "KB971033 x64 64 en-us", "", "Origin=Forward Inc |TrustLevel=5 |")

Note: This function must be called after calling the CreateDetectedSoftwarePatch function within each script. The CreateDetectedSoftwarePatchInstance function can be called as many times as the number of instances. Each instance gets a separate discovered software record.

Note: If two different Intellisigs detect the same software instance, two discovered software records are created, one for each Intellisig.

Input Parameters

This function has the following input parameters:

ProductName

Specifies the name of the product that the patch belongs to. You must have called the CreateDetectedSoftwareProduct function for this product within the script.

ProductVersionLabel

Specifies the version label of the product.

ReleaseName

Specifies the name of the release that the patch belongs to. You must have called the CreateDetectedSoftwareRelease function for this release within the script.

ReleaseVersionLabel

Specifies the version label of the release.

PatchName

Specifies the name of the detected software patch. You must have called the CreateDetectedSoftwarePatch function for this release within the script.

PatchVersionLabel

Specifies the version label of the patch.

OptionalProperties

Specifies the optional properties associated with the discovered software instance. Following optional properties are available for instances:

ProductGUID

Specifies the product UUID as detected by the script.

Label

Specifies a unique label for the instance.

Origin

Specifies the name of the Intellisig creator.

TrustLevel

Specifies the trust level of the creator of the Intellisig, script, or both.

InstallPath

Specifies the path to the product installation directory or the executable on the agent computer.

SerialNumber

Specifies the serial number of the instance.

LastAccessed

Specifies date and time when the instance was last accessed. Specify the value in the following format: yyyy-mm-dd:hr:mm. This value is assumed to be in local time.

Note: All other functions in dmscript that deal with time, operate in local time. Dmscript converts the time value to Unix format (seconds since 1-1-1970 UTC) before output. DSM Explorer displays this using the local time zone on the machine on which it is running.

CustomData

Specifies any other custom data you want store for the instance.

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_BADARGS

Indicates that the one or more mandatory parameters passed to the function are blank.

CASWDETECT_FILE_ERROR

Indicates that the function was unable to write to the file. See the log file for more information.

LogDetectedSoftwareError

The LogDetectedSoftwareError function writes error messages to an error log file. The function produces localizable error messages. Call this function for handling errors generated by create functions. The engine reads these error messages and displays in DSM Explorer.

Function format:

LogDetectedSoftwareError (MessageID as String, Properties as String)

Example:

```
LogDetectedSoftwareError("ISE:00400","PARAM5=Microsoft Windows 7 Ultimate x64 64 en-us|PARAM6=VersionNumber=6.1.7600 |VersionLabel=6.1.7600 |Language=en-us |Bitness=64 |Architecture=x64 |Manufacturer=Microsoft Corporation|Category=Operating Systems |Description=The Microsoft Windows 7 Product|PARAM7=SWDETECT_BADARGS");
```

The preceding example code adds the following message to the log file:
Intellisig Microsoft Windows 7 version 6.1.7600, UUID A7C1E14A-7C93-4E17-B4E5-45B796717F49, script win7.xml, Failed to create software product. The parameters were ProductName:Microsoft Windows 7 Ultimate x64 64 en-us OptionalProperties : VersionNumber=6.1.7600 |VersionLabel=6.1.7600 |Language=en-us |Bitness=64 |Architecture=x64 |Manufacturer=Microsoft Corporation|Category=Operating Systems |Description=The Microsoft Windows 7 Product. Return Code : SWDETECT_BADARGS

Input Parameters

This function has the following input parameters:

MessageID

Specifies the error code. For more information about available error codes, see Error Codes and Optional Properties.

Properties

Specifies a list of properties that are required to create a formatted message string. Specify the parameters passed to the respective create functions that you are handling. The parameters must match the error text. For the exact error text and function syntax, see Error Codes and Optional Properties. The error text uses PARAM1 to PARAM8 to create the message string. If the string does not contain param, value pairs, the string is displayed as is. The parameters PARAM1 to PARAM4 are automatically assigned to the following values:

- PARAM1 contains Intellisig UUID
- PARAM2 contains Intellisig name
- PARAM3 contains Intellisig version
- PARAM4 contains return code from the function

The values for parameters PARAM5 to PARAM8 differ depending on the create function you are handling. For example, if you are handling the failure of the CreateDetectedSoftwareProduct function, following are the error code and text:

Error code: ISE:00400

Error text: Intellisig %1$t version %2$t, UUID %3$t, script %4$t, Failed to create software product. The parameters were ProductName:%5$t OptionalProperties: %6$t. Return Code: %7$t

In this example, PARAM5 specifies the product name, PARAM6 specifies optional properties, and PARAM7 specifies the return code.

CloseDetectedSoftwareOutputFiles

The CloseDetectedSoftwareOutputFiles function closes the files opened by the OpenDetectedSoftwareOutputFiles function.

Function format:

```
CloseDetectedSoftwareOutputFiles() as integer
```

Return Values

CA_SWDETECT_OK

Indicates that the function completed the operation successfully.

CASWDETECT_FILEERROR

Indicates that the function was unable to close the file. See the log file for more information.

Additional Information on Intellisigs

Certificate used for Integrity Checking of Intellisigs

CA Client Automation maintains the integrity of an Intellisig by associating an encrypted hash with each script in the database. The agent verifies the hashes and then executes each Intellisig. If a hash fails verification, the agent declines to run the Intellisig and sends a message to the GUI indicating the verification error as well as the details of the specific Intellisigs that failed the integrity checks.

Note: The hash is encrypted using the certificate x509cert://dsm r11/cn=manager signer,o=computer associates, c=us tagged with ManagerSigner. If you use custom x509 certificates, Intellisigs automatically use the custom certificate for encryption.

Detect Software on a 64-bit OS Using Both 32-bit and 64-bit Registry

You can specify the architecture value of a registry when creating custom traditional signatures in DSM Explorer for both registry and file entries. The specified value checks that a binary is built for the specified Windows architecture.

Follow these steps:

1. Navigate to DSM Explorer, Software, Definitions, *software category*, Release of *software category*, Properties, Recognition, Advanced, Registry.
2. Select one of the following values in the Architecture drop-down box for a 64-bit OS:

   32

   Triggers a search in the 32-bit registry hives.

   64

   Triggers a search in the 64-bit registry hives.

Note: If you change the OS group type to UNIX, the Architecture drop-down box is disabled.

Set Default Timeout for Intellisigs

The default timeout of Intellisigs helps ensure that faulty Intellisigs do not run indefinitely on a computer. You can define a default timeout value for all the Intellisig scripts that are executed on a computer or for a specific Intellisig. You can apply a longer default time to agents where the load is high, for example, database servers, and where Intellisigs require extra time to complete. For desktop computers, where the load is low, you can configure a lower timeout value.

Follow these steps:

1. Navigate to DSM, Agent, Asset Management.
2. Define a value for the following parameter:

   IntellisigDefaultExecutionTimeout

   Specifies the default timeout value in minutes.

   Default: 5

Note: When you create an Intellisig, set its timeout value to default. This action helps ensure that the agent applies the configured default timeout. If the Intellisig has a timeout value greater than zero, the specified value is enforced when the Intellisig is run.

intellisigcmd - Command Line Tool

intellisigcmd is a command line tool for Intellisigs. This tool has the following format:

intellisigcmd <cmd> param1=value1 param2=value2 ... [<DB_Credentials>]

cmd

Specifies the import, export, or genuuid command.

DB_credentials

Specifies the database credentials of the MDB. By default, the credentials are retrieved from the comstore.

Use the following sample format to specify the DB credentials:

Example: SQLServer DB Credentials

dbvendor=mssql dbhost=myhost dbname=mdb dbuser=ca_itrm dbpassword=mypwd dbinstance=inst

Example: Oracle DB Credentials

dbvendor=oracle dbhost=myhost dbname=orcl dbuser=ca_itrm dbpassword=mypwd dbinstance=1521

intellisigcmd export—Export Intellisigs

The intellisigcmd export command lets you export Intellisigs. You can either use the DSM Explorer or the command to export Intellisigs.

Command format:

intellisigcmd export file=<export name> [type=xml|zip] [platform=all|windows|unix]

export name

Specifies the name of the Intellisig XML or zip file that you want to export. If you do not provide the file extension, and the type is xml, the command creates a folder with the given name.

type

Specifies whether you want to export an XML or zip file. If you do not include the type parameter, the command assumes the export type depending on the export file extension.

Valid values: xml, zip

platform

Specifies the platform for determining which Intellisigs are exported.

Valid values: all, windows, unix

Default: all

intellisigcmd import—Import Intellisigs

The intellisigcmd import command lets you import Intellisigs. You can either use the DSM Explorer or the command.

Command format:

intellisigcmd import [file=<import source>] [type=xml|zip] [mode=replace|mergenew|mergeall] [updateactive=yes|no] [delete=yes|no]

import source

Specifies the name of the XML or zip file to which you want to import the Intellisig. If you do not provide the file extension, the command assumes the file extension depending on the type.

Note: If you want to import to an XML file, verify that the supporting directories exist in the same folder as the XML file.

type

Specifies whether you want to import as XML or zip files. If you do not include this parameter, the command assumes the import type depending on the import file extension.

Valid values: xml, zip

mode

Specifies the import mode. Following import modes are supported:

Default: mergenew

replace

Replaces existing definitions with the definition being imported. Existing definitions are lost.

mergenew

Appends new Intellisig versions to the definitions on the manager. Existing definitions are not modified.

mergeall

Appends new Intellisig versions and updates the existing definitions that are included in the import file. Intellig versions that are not defined in the import files are not modified.

updateactive

Specifies whether active Intellisig versions can be updated during the import.

Valid values: Yes, Y, true, 1 or No, N, false, 0

Default: Yes

delete

Specifies whether you want to delete Intellisigs before the import. If you do not include the delete switch, none of the Intellisigs are deleted before import.

Default: No

intellisigcmd genuuid—Generate UUIDs

The intellisigcmd genuuid command lets you generate unique UUIDs which you can use when creating custom Intellisigs.

Command format:

intellisigcmd genuuid [num=<count>]

num

Specifies the number of UUIDs to be generated. If you do not specify this parameter, a single UUID is generated. Otherwise, <count> UUIDs are generated.
Valid Values: 1 to 1000

Display Functions

Display functions are not supported on Linux platforms.

ClearScreen or ClrScr - Clear the Screen

Valid on Windows platforms

The ClearScreen or ClrScr function clears the screen.

Function format:

```
ClearScreen() as Boolean
```

The display function always returns TRUE.

Example:

This example prints five lines, waits 5 seconds, and then clears the screen.

```
Dim counter as integer

'Print 5 lines
For counter = 1 to 5
 Print("Line " + str(counter))
Next

'Wait 5 seconds
Sleep(5000)

'Clear the print screen
ClearScreen()
Print("Screen Cleared")
```

InputBox - Create, Display, and Operate a Message Box Window

Valid on Windows and Windows CE

The InputBox function creates, displays, and operates a message box window. The message box contains an application-defined prompt, title, and a text box with an application-defined default value, and the OK and Cancel buttons.

This miscellaneous function has the format:

```
InputBox(
        prompt as String,
        title as String,
        default as String) as Boolean
```

prompt

Indicates the prompt to be displayed.

title

Specifies the message box title.

default

Specifies the default prompt input. If default is not a variable, a syntax error exists.

Note: The input box is NOT placed in the foreground.

If the OK button was selected, the function returns TRUE and copies the contents of the text box into the default variable. If the Cancel button was selected, the function returns FALSE and leaves the default variable unchanged.

Example:

This example demonstrates the use of InputBox.

```
Dim TargetPath, Question as string

TargetPath=WindowsDir

Question="Please specify the directory to use for installation"
if InputBox(Question,"Install",TargetPath) then

        ' Place code for actual installation here...

        MessageBox("Installation completed")
else
        MessageBox("Installation aborted")
End if

End:
```

MessageBox or MsgBox - Display a Message Box Window

Valid on Windows and Windows CE

The MessageBox or MsgBox function displays a message box window. The message box contains a message, a title, and any combination of the predefined buttons described by the style parameter.

Instead of MessageBox, you can use MsgBox to call the function.

Function format:

MessageBox(message as String, title as String, style as Integer) as Integer

MessageBox(message as String, title as String) as Integer

MessageBox(message as String, style as Integer) as Integer

MessageBox(message as String) as Integer

message

Identifies the string containing the message that is to be displayed.

title

Identifies an optional string containing the message box title.

Default: DMS

style

Identifies an optional integer specifying the contents and behavior of the message box. The style parameter can be any of the following predefined constants:

MB_OK

Adds box with button OK (Default)

MB_OKCANCEL

Adds box with buttons OK and Cancel

MB_YESNO

Adds box with buttons Yes and No

MB_RETRYCANCEL

Adds box with buttons Retry and Cancel

MB_YESNOCANCEL

Adds box with buttons Yes, No, and Cancel

MB_ABORTRETRYIGNORE

Adds box with buttons Abort, Retry, and Ignore

Constants for MessageBox Modality

By default, the user must respond to the message box before continuing work in the current window; however, the user can move to, and work in, windows of other applications.

MB_SYSTEMMODAL

All applications are suspended until the user responds to the message box. System modal message boxes are used to notify the user of serious, potentially damaging errors that require immediate attention.

Constants for MessageBox Icons

The default is as follows: No icon is shown.

MB_ICONEXCLAMATION

Adds an exclamation point icon appears in the message box.

MB_ICONINFORMATION

Adds an icon consisting of an "i" in a circle appears in the message box.

MB_ICONQUESTION

Adds a question-mark icon in the message box.

MB_ICONSTOP

Adds a stop sign icon (white "X" in a red circle) appears in the message box.

Constants for MessageBox Default Buttons

The default is as follows. The first button is the default.

MB_DEFBUTTON2

Makes the second button default.

MB_DEFBUTTON3

Makes the third button default.

Other Constants

MB_SETFOREGROUND

Sets the message box as the foreground window. If this is not coded, the current foreground window remains in the foreground.

You can combine each constant of one group with any constant of another group. For example, to display a message box with Abort, Retry, and Ignore (default) buttons, and a stop sign icon, and you want to suspend all applications until the user responds, code using the following style:

```
MB_ABORTRETRYIGNORE + MB_DEFBUTTON3 + MB_ICONSTOP + MB_SYSTEMMODAL
```

If the style parameter is omitted, MB_OK is assumed and the message box contains only the OK button.

If there is not enough memory to create the message box, the return value of the function is zero. Otherwise, it is one of the following button values returned by the message box:

IDOK

Value 1; the OK button was clicked.

IDCANCEL

Value 2; the Cancel button (or keyboard Esc key). was clicked.

IDABORT

Value 3; the Abort button was clicked.

IDRETRY

Value 4; the Retry button was clicked.

IDIGNORE

Value 5; the Ignore button was clicked.

IDYES

Value 6; the Yes button was clicked.

IDNO

Value 7; the No button was clicked.

Example:

This example checks with the user for creating a backup copy of the config.sys file, and creates the backup, if required.

```
Dim Src,Dst as string
Dim Question as string

Question="Do you want to make a backup of your config.sys?"

if MessageBox(Question,MB_YESNO)=IDYES then
        Src="C:\CONFIG.SYS"
        Dst="C:\CONFIG.BAK"
        if CopyFile(Src,Dst,TRUE) then MessageBox("CONFIG.BAK created.")
end if
```

ProgressBar - Set the Progress Bar

Valid on Windows platforms

The ProgressBar function sets the progress bar on the main window to the percentage indicated.

Display function format:

```
ProgressBar(percentage as Integer) as Boolean
```

percentage

Value of progress (percent).

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

This example shows the ProgressBar and ProgressText functions.

```
Dim counter as Integer
For counter = 1 TO 10
 ProgressBar(counter * 10)
 ProgressText("Now processing : " + Str(counter) )
 Sleep(500)                'Wait half second
Next
```

ProgressText - Copy a String to the Progress Text Field

Valid on Windows only

The ProgressText function copies the string to the progress text field in the Main window.

Display function format:

```
ProgressText(text as String) as Boolean
```

text

Defines text for the progress text field.

On successful completion, the function returns TRUE, otherwise it returns FALSE.

Example:

This example shows the ProgressBar and ProgressText functions.

```
Dim counter as Integer
For counter = 1 TO 10
 ProgressBar(counter * 10)
 ProgressText("Now processing : " + Str(counter) )
 Sleep(500)                'Wait half second
Next
```

Environment Functions

The Environment functions can be used on Windows NT/9x and UNIX platforms.

EnvExport - Export the Environment to a Text File

Valid on UNIX and Windows

The EnvExport function exports the current environment to a text file.

Function format:

```
EnvExport(filename as String) as Boolean
```

filename

Defines the name of the text file.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
Rem
Rem Exports defined environment variables
Rem
EnvExport("c:\temp\env.dat")
```

EnvGetFirst - Retrieve the Name of the First Environment Variable

Valid on UNIX and Windows

The EnvGetFirst function retrieves the first environment variable name.

Function format:

```
EnvGetFirst() as string
```

Defines a string containing the name of the first environment variable returned. If no more environment variables exist, an empty string is returned.

Example:

This example displays all the environment variable names.

```
Dim txt as String

txt = EnvGetFirst()
While txt <> ""
 Print(txt)
 txt = EnvGetNext()
Wend
```

EnvGetNext - Retrieve the Name of the Subsequent Environment Variables

Valid on UNIX and Windows

The EnvGetNext function continues the EnvGetFirst function, retrieving the name of subsequent environment variables. For each call of EnvGetNext, one environment variable name is returned until no more environment variables are found.

Function format:

```
EnvGetNext() as string
```

The return value is a string containing the name of an environment variable. If no more environment variables exist, an empty string is returned.

Example:

This example displays all the environment variable names

```
Dim txt as String

txt = EnvGetFirst()
While txt <> ""
 Print(txt)
 txt = EnvGetNext()
Wend
```

EnvGetString - Retrieve the Value of an Environment Variable

Valid on UNIX and Windows

The EnvGetString function retrieves the value of an environment variable.

Function format:

```
EnvGetString(name as string) as string
```

name

Defines the name of the environment variable.

The function returns a string containing the value associated with name. If an environment variable specified by name does not exist, the function returns an empty string.

Example:

This example displays the current search path.

```
Rem variable `MyPath´
Rem
Dim txt as String

txt = EnvGetString("PATH")

Print("The path: "+txt)
EnvSetString("MyPath",txt)
Print("My path: "+txt)
```

EnvImport - Import Environment Variables

Valid on UNIX and Windows

The EnvImport function imports environment variables from a text file.

Function format:

```
EnvImport(filename as String) as Boolean
```

filename

Defines the name of the file.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

This example imports additional env variables from a file.

```
EnvImport("c:\temp\env.dat")
```

C:\temp\env.dat contains for example:

```
Xantippe=greek
Yokohama=japanese
Fuel=explosive
```

File Content Functions

You can use the File Content functions on any platform.

CloseFile - Close the File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The CloseFile function closes the file associated with a file handle obtained from a call to OpenFile or CreateFile.

Function format:

```
CloseFile(handle as Integer) as Boolean
```

handle

File handle obtained from a call of OpenFile or CreateFile.

On successful completion, the function returns TRUE (non-zero). If the function fails (for example, when the handle is not the handle of a valid open file) the function returns FALSE.

Example:

This example creates a backup of the CONFIG.SYS file.

```
Dim fIn, fOut as integer   ' Declare file handles

Dim OneLine as string     ' String to hold one line


' First open the Input file...


fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

' ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")
        Goto End

End if

' ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

' ...Close Files, and signal success.

CloseFile(fIn)
CloseFile(fOut)
MessageBox("A backup of the CONFIG.SYS file was created","MyScript")


end:
```

CreatePipe - Create a Pipe

Valid on UNIX and Windows

Note: This function is new in CA Client Automation and does not work with older versions of the Script Interpreter.

The CreatePipe function creates a named pipe for reading or writing in server role.

Function format:

CreatePipe(pipename as String, access as Integer) as Integer

pipename

Specifies the name of the pipe to be created. On UNIX, pipename is a valid pathname.

On Windows it has the form:

\\.\pipe\xxx

where xxx can contain any characters except '\'.

access

Specifies in what direction the pipe is used.

O_READ

(value 0) Opened for reading.

O_WRITE

(value 1) Opened for writing.

The function returns a file handle that you can use with functions ReadFile, WriteFile, and CloseFile.

CreatePipe is used in the process that acts as a pipe server. A pipe client uses the OpenPipe function.

The pipe works in blocking mode. The read or write functions following CreatePipe wait until a client uses the inverse function.

Note: The function is not available on Windows 95/98/ME.

On successful completion, the function returns a non-negative integer, which is the file handle. If the function fails, it returns -1.

Example:

This example assumes that the example of OpenPipe is stored as file p0cl.dms.

```
rem example named pipe server
rem create two pipes
rem start client process
rem write command to one pipe
rem read response from second pipe

dim rc, h0, h1 as integer
dim pnam0, pnam1 as string
dim stopstring as string
dim s, cmd as string
dim iswin as boolean

if left(osstring, 3) = "Win" then
   pnam0 = "\\.\pipe\p0rsp"
   pnam1 = "\\.\pipe\p0cmd"
            iswin = 1
else
   pnam0 = "/tmp/p0rsp"
   pnam1 = "/tmp/p0cmd"
            iswin = 0
endif
stopstring = "---END---"

h0 = CreatePipe(pnam0, O_READ)            'read response form client
h1 = CreatePipe(pnam1, O_WRITE)           'send command to client
if h0 < 0 or h1           < 0 then
   print "create pipe failure. h0: " + str(h0) + "  h1: " + str(h1)
            exit
end if

if iswin then
            cmd = "dmscript p0cl.dms -o_dms: p0clout.txt -script "      + pnam1 + " " + pnam0
            rc = Exec(cmd, 0)
else
            cmd = "dmscript p0cl.dms -o_dms: p0clout.txt -script " + pnam1 + " " + pnam0 + " &"
            rc = Exec(cmd)
end if
print "exec rc: " + str(rc) + " cmd: " + cmd
```

```
rc = WriteFile(h1, "command")
if not(rc) then
        print "write command failed"
else
        while rc
                rc = ReadFile(h0, s)
                print "read response rc: " + str(rc) + " " + s
                if s = stopstring then
                        rc = 0
                        print "test successful"
                end if
        wend
end if

CloseFile(h0)
CloseFile(h1)
```

CreateFile - Create a New File

Valid on NetWare, Symbian OS, UNIX, Windows, and Windows CE

Note: On Windows 9x, do not use this function to create an .ini file. Windows OS creates the specified .ini file of a WriteIniSection() or WriteIniEntry().

The CreateFile function creates a file for writing.

Function format:

```
CreateFile(filename as String, mode as Integer) as Integer

CreateFile(filename as String) as Integer
```

filename

Specifies the name of the file.

mode

Specifies the mode in which to create the file. The value is one of the following predefined constants:

O_TEXT

(value 0, default) Text mode

O_BINARY

(value 1) Binary mode

If the mode parameter is omitted, text mode is assumed.

If the function succeeds, the return value is a non-negative integer, the file handle. If the function fails, it returns -1.

Example:

This example Creates a backup of the CONFIG.SYS file.

```
Dim fIn, fOut as integer   ' Declare file handles

Dim OneLine as string     ' String to hold one line


' First open the Input file...


fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

' ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")
        Goto End

End if

' ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

' ...Close Files, and signal success.

CloseFile(fIn)
CloseFile(fOut)
MessageBox("A backup of the CONFIG.SYS file was created","MyScript")


end:
```

EOF - Check for the End-of-File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The EOF function determines whether the end of the file has been reached.

Function format:

```
Eof(handle as Integer) as Boolean
```

handle

Identifies a file handle.

The function returns TRUE if the end of file has been reached; otherwise, it returns false.

Note: EOF returns TRUE when the last read operation has reached the end of file. The function does not check in advance to see whether a subsequent read command reached the end of a file.

Example:

This example creates a backup of the CONFIG.SYS file.

```
Dim fIn, fOut as integer   ' Declare file handles
Dim OneLine as string     ' String to hold one line

Rem First open the Input file...

fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

Rem ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")

CloseFile(fIn)
        Goto End
End if
```

```
Rem ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

Rem ...Close Files, and signal success.

CloseFile(fIn)
CloseFile(fOut)
MsgBox("A backup of the CONFIG.SYS file was created","MyScript",MB_OK)


end:
```

OpenFile - Open a File

Valid on NetWare, Symbian OS, UNIX, Windows, and Windows CE

The OpenFile function opens a file for reading or writing.

Function format:

```
OpenFile(filename as String, access as Integer, mode as Integer) as Integer

OpenFile(filename as String, access as Integer) as Integer
```

filename

Specifies the name of the file to open.

access

Specifies how to open the file. The access parameter can be any of the following predefined constants:

O_READ

(value 0) Opened for reading.

O_WRITE

(value 1) Created for writing. This overwrites the file if it exists.

O_APPEND

(value 2) Opened for writing at end of file. This creates a file if it does not exist.

O_UPDATE

(value 3) Opened for reading and writing.

mode

Indicates an optional parameter to specify binary mode. The mode parameter can be one of the following predefined constants:

O_TEXT

Value 0; text mode

O_BINARY

Value 1; binary mode

If the mode parameter is omitted, text mode is assumed.

If the function succeeds, the return value is a non-negative integer, the file handle. If the function fails, it returns -1.

Example:

This example creates a backup of the CONFIG.SYS file.

```
Dim fIn, fOut as integer   ' Declare file handles

Dim OneLine as string     ' String to hold one line


' First open the Input file...


fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

' ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")
        Goto End

End if

' ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

' ...Close Files, and signal success.
```

```
CloseFile(fIn)
CloseFile(fOut)
MessageBox("A backup of the CONFIG.SYS file was created","MyScript")

end:
```

OpenPipe - Open a Pipe

Valid on UNIX and Windows

Note: This function is new in CA Client Automation and does not work with older versions of the Script Interpreter.

The OpenPipe function opens a named pipe for reading or writing a client role.

Function format:

```
OpenPipe(pipename as String, access as Integer) as Integer
```

pipename

Specifies the name of the pipe to be opened.

On UNIX pipename is a valid pathname.

On Windows it has the form \\.\pipe*xxx* where *xxx* can contain any characters except '\'

access

Specifies in what direction the pipe is used.

O_READ

(value 0) Opened for reading.

O_WRITE

(value 1) Opened for writing.

You can use the returned file handle with the functions ReadFile, WriteFile, and CloseFile. OpenPipe is used in the process that acts as a pipe client. Before you call OpenPipe, call the CreatePipe.

The pipe works in blocking mode. The read or write functions following OpenPipe wait until the server uses the inverse function.

On successful completion the function returns a non-negative integer, which is the file handle. If the function fails, it returns -1.

Example:

This example shows a named pipe client.

```
dim rc, h0, h1, i as integer
dim pnam0, pnam1 as string
dim stopstring as string
dim s, s1 as string
dim iswin as boolean

pnam0 = argv(1)
pnam1 = argv(2)
stopstring = "---END---"

h0 = OpenPipe(pnam0, O_READ)
h1 = OpenPipe(pnam1, O_WRITE)
if h0 < 0 or h1          < 0 then
   print "open pipe failure. h0: " + str(h0) + "  h1: " + str(h1)
   print "pnam0: " + pnam0 + "  pnam1: " + pnam1
          exit
end if

rc = ReadFile(h0, s)
print "read command rc: " + str(rc) + " " + s
if rc then
          for i = 1 to 5
                    s1 = "RSP " + s + " line " + str(i)
                    rc = WriteFile(h1, s1)
                    if not(rc) then
                              print "write failed: " + s1
                              exit for
                    end if
          next i
          if rc then
                    rc = WriteFile(h1, stopstring)
                    if rc then
                       print "test successful"
                    end if
          end if
end if

CloseFile(h0)
CloseFile(h1)
```

ReadFile - Read Data from a File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The ReadFile function reads data from a file.

The file content function has *one* of the following formats:

ReadFile (handle as Integer, buffer as String) as Boolean

ReadFile (handle as Integer, buffer as String, buflen as Integer) as Boolean

ReadFile (handle as Integer, buffer as Void) as Boolean

ReadFile (handle as Integer, buffer as Void, buflen as Integer) as Boolean

handle

Identifies a file handle returned by a previous call of OpenFile or CreateFile.

buffer

Specifies a variable of any type.

Buffers of type arrays of char are treated as strings.

If the argument buffer is a string type variable, the function reads from the current file position and stops reading when it reads a newline character or the maximum number of characters are reached. If the argument buffer is any other type, the function reads from the current file position and stops when the buffer is full or the file pointer reaches end of file.

Void means that the type of object can be any valid type except for string and arrays of char.

buflen

Specifies the number of bytes to be read from the file into the buffer. If the file has insufficient bytes, all the bytes remaining are transmitted into the buffer.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Note: Do not mix the different formats of the ReadFile() function which can lead to unexpected results.

Unlike the other formats, the first format reads the string as a text line that is terminated by a newline sign or end of file and stores the information to the buffer without the newline sign. The interpreter maps this function to a different system read function than the other three formats. The other ReadFile formats are mapped to a common system read function.

The different systems read functions may use different cache and read pointers to retrieve the data. If you mix the formats, the data may not be read in the sequence in which it was stored on the disk.

In the case of the first format, the string read is a text line, terminated by a newline sign or end of file. The information is stored to the buffer without this newline. This is not true for the other ReadFile functions.

The interpreter maps this function to a different system read function than the other three formats, which are mapped to a common system read function. These different systems read functions also use different caches and read pointers to retrieve the data. If the formats are mixed, the data might not be read in the sequence in it was stored on the disk.

Example:

```
Dim fIn, fOut as integer   ' Declare file handles
Dim OneLine as string     ' String to hold one line

Rem First open the Input file...

fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

Rem ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")

CloseFile(fIn)
        Goto End
End if

Rem ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

Rem ...Close Files, and signal success.
```

```
CloseFile(fIn)
CloseFile(fOut)
MsgBox("A backup of the CONFIG.SYS file was created","MyScript",MB_OK)

end:
```

SeekFile - Reposition the Current Position in an Open File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The SeekFile function repositions the current position in an open file.

Function format:

```
Seekfile(handle as integer, position as integer) as Boolean
```

handle

Identifies a handle returned by a previous call.

position

Indicates the new position in the file. A position of 0 sets the position to the beginning of the file. The next read or write will be at the new position in the file.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
'       Update a file at the 4th double word. set it to 0.
Dim fHandle as integer

fHandle = OpenFile("out5.a", O_UPDATE)
if (fHandle = -1) then
        Print("OpenFile(""out5.a"", O_UPDATE) failed.")
        exit
endif
if Not(SeekFile(fHandle, 16)) then
        Print("SeekFile(fHandle, 16) failed.")
        exit
endif
if Not(WriteFile(fHandle, 0)) then
        Print("WriteFile(fHandle, 0) failed.")
        exit
endif
CloseFile(fHandle)
```

WriteFile - Write Data to a File

SValid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The WriteFile function writes data to a file.

Function formats:

WriteFile(handle as Integer, buffer as String) as Boolean

WriteFile(handle as Integer, buffer as String, buflen as Integer) as Boolean

WriteFile(handle as Integer, buffer as Void) as Boolean

WriteFile(handle as Integer, buffer as Void, buflen as Integer) as Boolean

handle

Identifies a file handle returned by OpenFile or CreateFile.

buffer

Indicates a variable of any type.

buflen

Specifies the number of characters to be stored in the file. If the number of char exceeds buflen, only the first buflen characters are stored in the file; otherwise, the whole buffer is stored.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Note: Do not mix the different formats of the WriteFile() function formats. Mixing the formats may lead to unexpected results.

The interpreter handles the first format differently; and when writing the string to disk, the interpreter always adds a newline sign.

The interpreter maps the first format to a different system write function than the other three formats, which are mapped to a common system write function. The different systems write functions may use different caches to store the data written. At close time, the caches are written to disk one after another. If you mix the formats, the data may not appear in the correct sequence.

The interpreter handles the following formats in a different way:

WriteFile(handle as Integer, buffer as String) as Boolean

When the string is written to disk, a newline is always added. This is not true for the other types of formats. The interpreter maps this function to a different system write function than the other three formats which were mapped to a common system write function. These different systems write functions may also use different caches to store the data written. At close time, the caches are written to disk one after another. If the formats are mixed, the data may not appear in the sequence - in which it was written. Therefore, do not mix this format with the other formats.

Example:

```
Dim fIn, fOut as integer   ' Declare file handles
Dim OneLine as string     ' String to hold one line

Rem First open the Input file...

fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

Rem ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")

CloseFile(fIn)
        Goto End
End if

Rem ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

Rem ...Close Files, and signal success.

CloseFile(fIn)
CloseFile(fOut)
MsgBox("A backup of the CONFIG.SYS file was created","MyScript",MB_OK)


end:
```

More Information:

OpenFile - Open a File (see page 62)
CreateFile - Create a New File (see page 59)

File System Functions

CopyFile or Copy Link - Copy an Existing File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The CopyFile or CopyLink function copies an existing file to a new file.

Function format:

```
CopyFile(
        source as String,
        target as String,
        overwrite as Boolean) as Boolean

CopyFile(source as String, target as String) as Boolean
```

source

Specifies the file name of an existing source file.

target

Identifies the name of the new file.

overwrite

Specifies if the function overwrites an existing target. The default, TRUE, overwrites an existing target. FALSE causes the function to fail if the target file already exists.

On completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

This example creates a backup directory, copies autoexec.bat to the directory, and renames it to autoexec.bat.

```
HELP_GETFILEATTRIBUTES

If Not(ExistDirectory("c:\backup")) then CreateDirectory("c:\backup")
CopyFile("c:\autoexec.bat", "c:\backup\autoexec.bat")
RenameFile("c:\backup\autoexec.bat", "c:\backup\autoexec.bak")
```

CopyTree - Copy a Directory Tree

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The CopyTree function copies a directory tree. If the target path does not exist, this function creates it.

Function format:

```
CopyTree(
        source as String,
        target as String,
        overwrite as Boolean) as Boolean

CopyTree(source as String, target as String) as Boolean
```

source

Indicates the name of an existing directory.

target

Indicates the name of the new directory.

overwrite

Specifies if the function overwrites an existing target.

The default, TRUE, overwrites an existing target. FALSE causes the function to fail if the target file already exists.

Example:

This example copies the clientws directory tree to CliWS.

```
CopyTree("c:\clientws", "c:\cliws")
```

On successful completion, the function returns TRUE; otherwise, returns FALSE.

CreateDirectory or MkDir - Create a New Directory

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The CreateDirectory or MkDir function creates a new directory.

Function format:

```
MkDir(pathstring as String) as Boolean

CreateDirectory(pathstring as String) as Boolean
```

pathstring

Specifies the name of the directory to create.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

This example creates a backup directory, copies autoexec.bat to the directory, and renames it to autoexec.bat.

```
HELP_GETFILEATTRIBUTES

If Not(ExistDirectory("c:\backup")) then CreateDirectory("c:\backup")
CopyFile("c:\autoexec.bat", "c:\backup\autoexec.bat")
RenameFile("c:\backup\autoexec.bat", "c:\backup\autoexec.bak")
```

CreateFile - Create a New File

Valid on NetWare, Symbian OS, UNIX, Windows, and Windows CE

Note: On Windows 9x, do not use this function to create an .ini file. Windows OS creates the specified .ini file of a WriteIniSection() or WriteIniEntry().

The CreateFile function creates a file for writing.

Function format:

```
CreateFile(filename as String, mode as Integer) as Integer

CreateFile(filename as String) as Integer
```

filename

Specifies the name of the file.

mode

Specifies the mode in which to create the file. The value is one of the following predefined constants:

O_TEXT

(value 0, default) Text mode

O_BINARY

(value 1) Binary mode

If the mode parameter is omitted, text mode is assumed.

If the function succeeds, the return value is a non-negative integer, the file handle. If the function fails, it returns -1.

Example:

This example Creates a backup of the CONFIG.SYS file.

```
Dim fIn, fOut as integer   ' Declare file handles

Dim OneLine as string     ' String to hold one line


' First open the Input file...


fIn=OpenFile("C:\CONFIG.SYS",O_READ)
if fIn<0 then
        MessageBox("Unable to open input file","Error")
        Goto End
End if

' ...Then create the output file...

fOut=CreateFile("C:\CONFIG.BAK")
if fOut<0 then
        MessageBox("Unable to create output file","Error")
        Goto End

End if

' ...Copy lines until none left...

while Not(Eof(fIn))
        if ReadFile(fIn,OneLine) then WriteFile(fOut,OneLine)
wend

' ...Close Files, and signal success.

CloseFile(fIn)
CloseFile(fOut)
MessageBox("A backup of the CONFIG.SYS file was created","MyScript")


end:
```

DeleteFile or DelFile - Delete a File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The DeleteFile or DelFile function removes the specified file.

Function format:

```
DeleteFile(filename as String) as Boolean

DelFile(filename as String) as Boolean

DeleteFile(filename as String, mode as Integer) as Boolean

DelFile(filename as String, mode as Integer) as Boolean
```

filename

Indicates the file name and path.

mode

Use on Windows only. The mode values are:

0 Delete file if not read-only

1 Delete file even if read-only

2 Deletion is delayed until next reboot

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
Rem
Rem By deleting the file Ncclient.ini a new workstation name detect
Rem will be forced next time the Unicenter Asset Management agent is executed.
Rem
DeleteFile(ComputerPath+"Ncclient.ini")
Rem ComnputerPath is an Unicenter Asset Management constant
```

DeleteTree or DelTree - Remove a Directory Tree

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The DeleteTree or DelTree function removes the specified directory tree including all of its subdirectories, even if they are not empty.

Note: The specified directory must not be write-protected.

Function format:

DeleteTree(pathname as String) as Boolean

DelTree(pathname as String) as Boolean

pathname

Specifies the path of the tree that is to be deleted.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
Rem
Rem By deleting the directory bak this forces a full collect next time
Rem the Unicenter Asset Management agent is executed.
Rem
DeleteTree(ComputerPath+"bak")
Rem ComputerPath is a Unicenter Asset Management constant
```

ExistDir or ExistDirectory - Check if a Directory Exists

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The ExistDir or ExistDirectory function checks whether a directory exists.

Function format:

ExistDir(pathname as String) as Boolean

ExistDirectory(pathname as String) as Boolean

pathname

Identifies the directory path.

If the directory exists, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
HELP_GETFILEATTRIBUTES

If Not(ExistDirectory("c:\backup")) then CreateDirectory("c:\backup")
CopyFile("c:\autoexec.bat", "c:\backup\autoexec.bat")
RenameFile("c:\backup\autoexec.bat", "c:\backup\autoexec.bak")
```

ExistFile - Check if a File Exists

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The ExistFile function checks for the existence of a file.

Function format:

ExistFile(filename as String) as Boolean

filename

Identifies the file path.

If the file exists, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
Dim file, section, entry, value as string

Dim LF, CR as char
Dim rtr as integer

ClrScr()
LF = 0x0a
CR = 0x0d
file = "c:\dmscript.ini"
if Not(ExistFile(file)) then
        rtr = CreateFile(file, O_TEXT)
        if rtr = -1 then
                SetStatus(1)
                exit
        end if
        closeFile(rtr)
end if

section = "Section 1"
value = "Param_1 = Wert_1" + LF + "Param_2 = Wert_2"

if WriteIniSection( section, value, file) then
        Print("WriteIniSection successfully completed.")
else
        Print("WriteIniSection failed.")
endif
```

```
section = "Section 2"
entry = "Param_3"
value = "Wert_3"

if WriteIniEntry( section, entry, value, file) then
        Print("WriteIniFile successfully completed.")
else
        Print("WriteIniFile failed.")
endif

section = "Section 1"
rtr = ReadIniSection(section, value, file)
if (rtr > 0) then
        Print( Str(rtr) + CR + LF + value)
else
        Print("ReadIniSection failed.")
endif

section = "Section 2"
entry = "Param_3"
rtr = ReadIniEntry(section, entry, value, file)

if (rtr > 0) then
        Print( Str(rtr) + CR + LF + entry + " = " + value)
else
        Print("ReadIniEntry failed.")

endif
```

FindClose - Close a Find Operation

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The FindClose function closes a find operation.

Function format:

```
FindClose(Handle as Integer) as Boolean
```

handle

Identifies the handle of the find operation to be closed.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

```
Function contents(root as string) As Boolean
        Dim fileName As String
        Dim iHandle As Integer
        Dim rootRoot As String

        rootRoot = Mid(root, 0, len(root)-1)
        iHandle = FindFirstFile(root, fileName)) Then
        If iHandle = 0 Then
                Print("No files found at '""" + root + """")
        Else
                Repeat
                        If Is_Dir(rootRoot + fileName) Then
                                Print("""" + rootRoot + fileName + '"" is a directory")
                                if Not(fileName = ".") AND Not(fileName = "..") Then
                                        contents(rootRoot + fileName + "\*")
                                End If
                        Else
                                Print("""" + rootRoot + fileName + '"" is a file")
                        End If
                Until Not(FindNextFile(iHandle, fileName))

                FindClose(iHandle)
        End If
        contents = TRUE
End Function

Dim i As Integer
ClrScr()
if Argc() = 1 Then
        Exit
End If
For i=1 To Argc()-1
        Print(NEWLINE$ + "Recursive List of '""" + Argv(i) + """")
        Contents(Argv(i))
Next i
```

Invokation: dmscript sample.dms c:* -w_dms

More Information:

FindFirstFile - Search for Files in a Directory (see page 81)

FindFirst - Return the Name of the First File in the Search

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The FindFirst function returns the name of the first file that matches a specified filter.

Function format:

FindFirst(Filter as string, Attrib as integer) as string

FindFirst(Filter as string) as string

Filter

Specifies the files to include in the scan. This parameter can contain a drive specification, a path, and a file name for the files to be found. The file name can contain wildcard characters (? or *).

Attrib

Specifies the file attribute for selecting eligible files in the scan. If the Attrib parameter is omitted, FA_ALLFILES is assumed. Attrib can be selected by ORing the following values:

FA_NONE

Normal file, no attributes

FA_RDONLY

Read-only files

FA_HIDDEN

Hidden files

FA_SYSTEM

System files

FA_LABEL

Volume label

FA_DIR

Directory

FA_ARCHIVE

Archive files

FA_ALLFILES

All files

FA_ANYFILE

Any file

On successful completion, the FindFirst function returns a string containing the file name of the first matching file. Otherwise, it returns an empty string.

Example:

```
Dim Filename as string

Filename=FindFirst("C:\*.BAT",FA_ANYFILE)
while Filename<>""
        print Filename
        Filename=FindNext()
wend
```

FindFirstFile - Search for Files in a Directory

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The FindFirstFile function searches for files in a specified directory.

Function format:

```
FindFirstFile(namespec as String, filename as String) as Integer
```

namespec

Identifies the directory path or the file path with wildcard file specification. Do not use a literal string. For example,

NOT:

```
Dim filename as string
Dim findHandle as integer
findHandle=FindFirstFile("c:\windows\*.dll",filename).
```

BUT:

```
Dim filename as string
Dim filehandle as integer
Dim path as string
Path="c:\windows\*.dll"
Filehandle=FindFirstFile(path,filename)
```

filename

An output parameter that contains the name of the file that has been found.

On successful completion, the function returns a find handle that you use as input to FindNextFile and FindClose; otherwise, the function returns 0. If the function fails it returns 0.

Example:

This example recursively lists the contents of directories passed as arguments.

```
Function contents(root as string) As Boolean
        Dim fileName As String
        Dim iHandle As Integer
        Dim rootRoot As String

        rootRoot = Mid(root, 0, len(root)-1)
        iHandle = FindFirstFile(root, fileName)) Then
        If iHandle = 0 Then
                Print("No files found at '""" + root + """")
        Else
                Repeat
                        If Is_Dir(rootRoot + fileName) Then
                                Print("""" + rootRoot + fileName + '"" is a directory")
                                if Not(fileName = ".") AND Not(fileName = "..") Then
                                        contents(rootRoot + fileName + "\*")
                                End If
                        Else
                                Print("""" + rootRoot + fileName + '"" is a file")
                        End If
                Until Not(FindNextFile(iHandle, fileName))

                FindClose(iHandle)
        End If
        contents = TRUE
End Function

Dim i As Integer
ClrScr()
if Argc() = 1 Then
        Exit
End If
For i=1 To Argc()-1
        Print(NEWLINE$ + "Recursive List of '""" + Argv(i) + """")
        Contents(Argv(i))
Next i
```

Invokation: dmscript sample.dms c:* -w_dms

FindNext

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The FindNext function continues a directory scan, which you use to fetch subsequent files that match the criteria given in FindFirst. FindNext returns one file name for each call of FindNext until no more matching files are found in the directory.

Function format:

```
FindNext() as string
```

Each call of FindNext returns one file name. When no more matching files are found in the directory, it returns an empty string ("").

Example:

```
Dim Filename as string

Filename=FindFirst("C:\*.BAT",FA_ANYFILE)
while Filename<>""
        print Filename
        Filename=FindNext()
wend
```

FindNextFile - Continue FindFirstFile Directory Scan

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The FindNextFile function continues a directory scan.

Function format:

```
FindNextFile(findHandle as Integer, filename as String) as Integer
```

findHandle

Identifies the handle returned from FindFirstFile.

filename

Identifies an output parameter that contains the name of the file that has been found.

The function returns a value of 1 if a file was found and a return value of 0 if no more files can be found.

Example:

```
Function contents(root as string) As Boolean
        Dim fileName As String
        Dim iHandle As Integer
        Dim rootRoot As String

        rootRoot = Mid(root, 0, len(root)-1)
        iHandle = FindFirstFile(root, fileName)) Then
        If iHandle = 0 Then
                Print("No files found at '"" + root + ""'")
        Else
                Repeat
                        If Is_Dir(rootRoot + fileName) Then
                                Print("""" + rootRoot + fileName + '"" is a directory")
                                if Not(fileName = ".") AND Not(fileName = "..") Then
                                        contents(rootRoot + fileName + "\*")
                                End If
                        Else
                                Print("""" + rootRoot + fileName + '"" is a file")
                        End If
                Until Not(FindNextFile(iHandle, fileName))

                FindClose(iHandle)
        End If
        contents = TRUE
End Function

Dim i As Integer
ClrScr()
if Argc() = 1 Then
        Exit
End If
For i=1 To Argc()-1
        Print(NEWLINE$ + "Recursive List of '"" + Argv(i) + ""'")
        Contents(Argv(i))
Next i
```

Invokation: dmscript sample.dms c:* -w_dms

GetDir or GetDirectory - Get the Name of the Current Directory

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The GetDir or GetDirectory function gets the name of the current directory.

Function format:

GetDir(n1 as Integer) as String

GetDirectory(n1 as Integer) as String

n1

Indicates the drive number.

On successful completion, the function returns a string containing the name of the current directory for the specified drive. Otherwise, it returns an empty string.

The function returns the name of the current working directory for the drive indicated by the parameter n1 (which specifies a drive number: 0 for default, 1 for A, 2 for B, and so on).

Note: On UNIX platforms "n1" has no meaning.

Example:

This example creates a directory, sets the active directory to it, and gets the name the name of the directory. The example sets the default directory to C:\ and changes the name of the directory created. If the directory exists (that is, it was not renamed), it is deleted.

```
'Create dir
CreateDirectory("C:\AMTest")

'Set active dir
SetDirectory("C:\AMTest")
MessageBox(" Current Directory is : " + GetDirectory(0) )

'Set Default to C:\
SetDirectory("C:\")

'Rename the directory
RenameDirectory("C:\AMTest","C:\AMTest_1")
```

```
If ExistDirectory("C:\AMTest_1") THEN
 RemoveDirectory("C:\AMTest_1")

Else
 MessageBox("Could not find directory : C:\AMTest_1")
Endif
```

GetDiskFree, GetDiskFreeB, GetDiskFreeKB, and GetDiskFreeMB - Return Free Disk Space

Valid on Symbian OS, Windows and Windows CE.

Function format:

```
GetDiskFree(drivenumb1 as Integer) as Integer

GetDiskFreeB(drivenumb1 as Integer) as Integer

GetDiskFreeKB(drivenumb1 as Integer) as Integer

GetDiskFreeMB(drivenumb1 as Integer) as Integer
```

drivenumb1

Indicates the drive number. Use 0 for the default, 1 for A, 2 for B, and so on.

The functions return the size in a 32-bit integer format. Large disks may exceed the capacity of such a representation, resulting in wrong sizes or negative values due to truncation.

Each function returns an integer, which is the free disk space, as follows:

GetDiskFree

Free disk space value in KB.

GetDiskFreeB

Free disk space value in bytes.

GetDiskFreeKB

Free disk space value in KB.

GetDiskFreeMB

Free disk space value in MB.

Note: The GetFileStoreFree function is portable and should be used in preference to GetDiskFree functions.

Example:

```
Rem

Rem This displays all information about the current drive in KiloBytes
Rem
Dim drive AS Integer

drive = GetDrive()

Print("Current drive is  : " + Str( drive ) )
Print("Current drive Size : " + Str( GetDiskSizeKB( drive )) + " KB"))
Print("Current drive Free : " + Str( GetDiskFreeKB( drive ))
```

GetDiskSize, GetDiskSizeB, GetDiskSizeKB, and GetDiskSizeMB - Return the Disk Size

Valid on Symbian OS, Windows, and Windows CE

The GetDiskSize function returns the disk size in KB, the GetDiskSizeB function returns the disk size in bytes, the GetDiskSizeKB function returns the disk size in KB, and the GetDiskSizeMB function returns the disk size in MB.

Function format:

```
GetDiskSize(drivenumb1 as Integer) as Integer

GetDiskSizeB(drivenumb1 as Integer) as Integer

GetDiskSizeKB(drivenumb1 as Integer) as Integer

GetDiskSizeMB(drivenumb1 as Integer) as Integer
```

drivenumb1

Indicates the drive number. Use 0 for the default, 1 for A, 2 for B, and so on.

The functions return the size in a 32-bit integer format. Large disks may exceed the capacity of such a representation, resulting in wrong sizes or negative values due to truncation.

Each function returns an integer, which is the disk size, as follows:

GetDiskSize

Disk size value in KB.

GetDiskSizeB

Disk size space value in bytes.

GetDiskSizeKB

Disk size value in KB.

GetDiskSizeMB

Disk size value in MB.

Note: Use the GetFileStoreSize function, which is portable, instead of the GetDiskSize functions.

Example:

```
Dim Size,Used,Free as integer
Dim out as string

Size=GetDiskSize(3)
if (Size<>-1) then
        out="Total size: "+chr(9)+str(Size)+" bytes"+ chr(10)
        Used=GetDiskUsage(3)
        out=out+"Bytes used: "+chr(9)+str(Used)+" bytes"+ chr(10)
        Free=GetDiskFre
```

GetDiskUsage, GetDiskUsageB, GetDiskUsageKB, and GetDiskUsageMB - Return the Amount of Disk Usage

Valid on Symbian OS, Windows, and Windows CE

The GetDiskUsage and the GetDiskUsageKB functions return the amount of disk usage in KB, the GetDiskUsageB function returns the amount of disk usage in bytes, and the GetDiskUsageMB function returns the amount of disk usage in MB.

Function format:

```
GetDiskUsage(drivenumb1 as Integer) as Integer

GetDiskUsageB(drivenumb1 as Integer) as Integer

GetDiskUsageKB(drivenumb1 as Integer) as Integer

GetDiskUsageMB(drivenumb1 as Integer) as Integer
```

drivenumb1

Indicates the drive number.

The functions return the size in a 32-bit integer format. Large disks may exceed the capacity, resulting in wrong sizes or negative values due to truncation.

The drive is indicated by the drivenumb1 parameter, which specifies a drive number: 0 for the default, 1 for A, 2 for B, and so on.

Each function returns an integer, which is the disk usage, as follows:

GetDiskUsage

Indicates the disk usage value in KB.

GetDiskUsageB

Indicates the disk usage value in bytes.

GetDiskUsageKB

Indicates the disk usage value in KB.

GetDiskUsageMB

Indicates the disk usage value in MB.

Note: Use the GetFileStore function instead as it is portable.

Example:

```
Dim drive AS Integer
drive = GetDrive()
Print("Current drive is  : " + Str( drive ))
Print("Current drive Size : " + Str( GetDiskSizeMB( drive )) + " MB"))
Print("Current drive Free : " + Str( GetDiskFreeMB( drive )) +
```

GetDrive - Return the Current Drive Number

Valid on Symbian OS and Windows

The GetDrive function returns the current drive number: 1 for A, 2 for B, 3 for C, and so on.

Function format:

```
GetDrive() as Integer
```

The function returns the current drive number: 1 for A, 2 for B, 3 for C, and so on.

Example:

```
Dim drive AS Integer

drive = GetDrive()

Print("Current drive is : " + Str( drive ) )
Print("Current drive Size : " + Str( GetDiskSize( drive )) )
Print("Current drive Free : " + Str( GetDiskFree( drive )) )
Print("Current drive Usage: " + Str( GetDiskUsage( drive )) )
```

GetFileAttributes - Return the File Attributes

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The GetFileAttributes function retrieves the attributes of the file or directory indicated by file name.

Function format:

```
GetFileAttributes(filename as String) as Integer
```

filename

Identifies the file path.

The following list indicates the value of the file attributes for all supported Windows platforms. Parameter values are decimal.

READONLY

Value: 1

HIDDEN

Value: 2

SYSTEM

Value: 4

DIRECTORY

Value: 16

ARCHIVE

Value: 32

NORMAL

Value: 128

TEMPORARY

Value: 256

COMPRESSED

Value: 2048

OFFLINE

Value: 4096

On successful completion, the function returns a value containing the attributes of the specified file or directory. If the file does not exist, the function returns the value -1.

Example:

This example reports the attributes on AUTOEXEC.BAT and switches the Hidden flag.

```
dim attr as integer
dim out as string

attr=GetFileAttributes("C:BAT")
if (attr<>-1) then
        out=out+"Attributes: "+chr(9)+"["
        if attr and FA_ARCHIVE then out=out+"A"
        if attr and FA_SYSTEM then out=out+"S"
        if attr and FA_HIDDEN then out=out+"H"
        if attr and FA_RDONLY then out=out+"R"

        out=out+"]"
        MsgBox("Information on C:\AUTOEXEC.BAT"+chr(10)+chr(10)+out)
else
        MsgBox("Cannot find C:\AUTOEXEC.BAT")
end if
if attr and FA_HIDDEN then
        attr = attr – FA_HIDDEN
        SetFileAttributes("C:\AUTOEXEC.BAT", attr)
else
        attr = attr + FA_HIDDEN
        SetFileAttributes("C:\AUTOEXEC.BAT", attr)
End If
end:
```

GetFileSize - Return the Size of the File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The GetFileSize function provides the size of the file in bytes.

Function format:

GetFileSize(filename as String) as Integer

filename

Identifies the path to the file.

The return value is the amount in bytes.

Example:

This example reports the size and creation date of the file AUTOEXEC.BAT.

```
dim size as integer
dim time as DateTime
dim out as string

size=GetFileSize("C:\AUTOEXEC.BAT")
if (size<>-1) then
        out="File size: "+chr(9)+chr(9)+str(size)+" bytes"+ chr(10)
        time=GetFileTime("C:\AUTOEXEC.BAT")
        out=out+"Creation date: "+chr(9)+str(time.month)+"/"+str(time.day)+"/"+str(time.year)
        out=out+" "+str(time.hour)+":"+str(time.minute)+" "
        MsgBox("Information on C:\AUTOEXEC.BAT"+chr(10)+chr(10)+out)

else
        MsgBox("Cannot find C:\AUTOEXEC.BAT")
end if
end:
```

GetFileStoreFree, GetFileStoreSize, and GetFileStoreUsage - Return the Free Space, Total Space, and Used Space

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The GetFileStoreFree, GetFileStoreSize, and GetFileStoreUsage functions provide the free space, total space, and used space on the file system in KB, respectively.

Functions format:

```
GetFileStoreFree(filename as String) as Integer

GetFileStoreSize(filename as String) as Integer

GetFileStoreUsage(filename as String) as Integer
```

filename

Specifies a path to a file or directory and identifies the file system. A file system has the usual meaning in UNIX. In Windows it is a drive.

Each function returns the amount in KB.

Example:

This example lists the size, and the used and free space of drive C:\.

```
Dim size, free, used As Integer
Dim TAB As Char
TAB = 0x0009
ClrScr()

size = GetFileStoreSize("c:\")
free = GetFileStoreFree("c:\")
used = GetFileStoreUsage("c:\")
Print("Capacity of drive c:\")
Print("Total size" + TAB + TAB + Str(size))
Print("Free storage" + TAB + Str(free))
Print("Used storage" + TAB + Str(used))
```

GetFileTime - Return File Creation Date, Last Access Date, or Last Modified Date

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The GetFileTime function returns the creation date of a file (format 1), or, if the type of time is specified (format 2), the creation date, or last access date, or last modification date.

Function format:

```
GetFileTime(filename as String) as DateTime

GetFileTime(filename as String, type as Integer) as DateTime
```

filename

Specifies the name of the file to investigate.

type

Type of file time to retrieve.

The following values are valid:

0 Date and time of creation.

1 Date and time of last access.

2 Date and time of last modification (Default).

The DateTime structure is defined as follows:

```
Type DateTime
    Year as Integer
    Month as Integer
    Day as Integer
    DayOfWeek as Integer
    Hour as Integer
    Minute as Integer
    Second as Integer
End Type
```

On successful completion, the function returns the creation date and time of a file (format 1), or the time specified in type (format 2). If the file does not exist, it returns a DateTime structure with all members set to -1.

Example:

```
dim size as integer
dim time as DateTime
dim out as string

size=GetFileSize("C:\AUTOEXEC.BAT")
if (size<>-1) then
        out="File size: "+chr(9)+chr(9)+str(size)+" bytes"+ chr(10)
        time=GetFileTime("C:\AUTOEXEC.BAT")
        out=out+"Creation date: "+chr(9)+str(time.month)+"/"+str(time.day)+"/"+str(time.year)
        out=out+" "+str(time.hour)+":"+str(time.minute)+" "
        MsgBox("Information on C:\AUTOEXEC.BAT"+chr(10)+chr(10)+out)

else
        MsgBox("Cannot find C:\AUTOEXEC.BAT")
end if
end:
```

GetLongFileName - Convert Short File Name to Long File Name

Valid on Windows

The GetLongFileName function converts the short file name to the long file name.

Function format:

GetLongFileName(shortFileName as string, longFileName as string) as Boolean

shortFileName

An input parameter. It is the MS DOS 8.3 file name to be expanded into a long file name format.

longFileName

An output parameter and receives the long, expanded file name.

On successful execution, the function returns TRUE; otherwise, it returns FALSE. To convert a file name, the related file must exist on the system or the function fails.

Example:

This example converts a long file name into a short one and converts it back.

```
Dim lName, oName, sName As String
ClrScr()

oName ="c:\temp\a bb ccc dddd\abcdefghijklmnopqrstuvwxyz.txt"
If Not(GetShortFileName(oName, sName)) Then
        MessageBox("Can not convert to short", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(1)
        Exit
End If
Print( oName + " - " + sName)
If Not(GetLongFileName(sName, lName)) Then
        MessageBox("Can not convert to long", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(2)
        Exit
End If
Print( sName + " - " + lName)
```

GetShortFileName - Convert Long File Name to Short File Name

Valid on Symbian OS and Windows

The GetShortFileName function converts the long file to the short file name.

Function format:

GetShortFileName(longFileName as string, shortFileName as string) as Boolean

longFileName

This is an input parameter. It is the file name to be converted into a short file name (8.3 notation).

shortFileName

This is an output parameter and receives the converted file name.

On successful execution, this function returns TRUE; otherwise, it returns FALSE. To convert a file name the related file must exist on the system or the function fails.

Example:

This example converts a long file name into a short one and converts it back.

```
Dim lName, oName, sName As String
ClrScr()

oName ="c:\temp\a bb ccc dddd\abcdefghijklmnopqrstuvwxyz.txt"
If Not(GetShortFileName(oName, sName)) Then
        MessageBox("Can not convert to short", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(1)
        Exit
End If
Print( oName + " - " + sName)
If Not(GetLongFileName(sName, lName)) Then
        MessageBox("Can not convert to long", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(2)
        Exit
End If

Print( sName + " - " + lName)
```

GetSpecialDir - Get the Special Directory Name and Path

Valid on Windows and Windows CE

The GetSpecialDir function gets the name and path of special directories.

Function format:

```
GetSpecialDir(Id as integer, Path as string) as integer
```

Id

The identifier of the special directory.

Path

Drive and path name of the specified directory (Output parameter).

On successful completion, the function returns zero (0); otherwise, it returns a non-zero value.

Note: If the location specified by the CSIDL parameter is not part of the file system, this function fails.

Example:

```
Rem sample for special directories

Dim dirName As String
ClrScr()

If GetSpecialDir(CSIDL_APPDATA, dirName) <> 0 Then
        MsgBox("Retrieve CSIDL_APPDATA failed!", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(1)
        Exit
End If

Print("CSIDL_APPDATA: '" + dirName + "'")
```

Is_Archive - Check if the File is Archived

Valid on NetWare, Windows, and Windows CE

The Is_Archive function checks the file for the specified attribute.

Function format:

```
Is_Archive(filename as String) as Boolean
```

filename

Identifies a file path.

The function returns TRUE if the file is archived; otherwise, it returns FALSE.

Example:

```
If Is_Archive(Argv(1)) Then
        print(Argv(1) + '' indicates 'Archive'.'')
Else
        print(Argv(1) + '' does not indicate 'Archive'.'')
End If
```

Is_Blk - Check if the File is a Block Special File

Valid on UNIX only

The Is_Blk function checks the file for the specified attribute.

Function format:

```
Is_Blk(filename as String) as Boolean
```

filename

Identifies a file path.

Returns TRUE if the file is a block special file, and returns FALSE in all other cases.

Example:

```
If Is_Blk(Argv(1)) Then
        print(Argv(1) + '' indicates 'Blk'.'')
Else
        print(Argv(1) + '' does not indicate 'Blk'.'')
End If
```

Is_Chr - Check if the File is a Character Special File

Valid on UNIX only

The Is_Chr function checks the file for the specified attribute.

Function format:

```
Is_Chr(filename as String) as Boolean
```

filename

Identifies a file path.

Returns TRUE if the file is a character special file and FALSE otherwise.

Example:

```
If Is_Chr(Argv(1)) Then
        print(Argv(1) + " indicates 'Chr'.")
Else
        print(Argv(1) + " does not indicate 'Chr'.")
End If
```

Is_Dir - Check if the File is a Directory

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Is_Dir function checks the file for the specified attribute.

Function format:

```
Is_Dir(filename as String) as Boolean
```

filename

Identifies a file path.

Use the correct syntax for file names and directory names. For example, C:\WINNT is a file name, not a directory name. The syntax for a directory is C:\WINNT*.

This function returns TRUE if the file is a directory; otherwise, it returns FALSE.

Example:

```
If Is_Dir(Argv(1)) Then
        print(Argv(1) + " indicates 'Dir'.")
Else
        print(Argv(1) + " does not indicate 'Dir'.")
End If
```

Is_Fifo - Check the File for the 'Pipe Special' Attribute

Valid on UNIX only

The Is_Fifo function checks the file for the specified attribute.

Function format:

```
Is_Fifo(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the file is a pipe special file; otherwise, the function returns FALSE.

Example:

```
If Is_Fifo(Argv(1)) Then
        print(Argv(1) + " indicates 'Fifo'.")
Else
        print(Argv(1) + " does not indicate 'Fifo'.")
End If
```

Is_File - Check the File is a Regular File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE.

The Is_File function checks the file for the specified attribute.

Function format:

```
Is_File(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the file is a regular file; otherwise, returns FALSE.

Example:

```
If Is_File(Argv(1)) Then
        print(Argv(1) + " indicates 'File'.")
Else
        print(Argv(1) + " does not indicate 'File'.")
End If
```

Is_G_R - Check the File for the 'Group Read' Attribute

Valid on UNIX only

The Is_G_R function checks the file for the specified attribute Group Read.

Function format:

```
Is_G_R(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set and FALSE otherwise.

Example:

```
If Is_G_R(Argv(1)) Then
        print(Argv(1) + " indicates 'G_R'.")
Else
        print(Argv(1) + " does not indicate 'G_R'.")
End If
```

Is_G_W - Check the File for the 'Group Write' Attribute

Valid on UNIX only

The Is_G_W function checks the file for the specified attribute Group Write.

Function format:

```
Is_G_W(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_G_W(Argv(1)) Then
        print(Argv(1) + " indicates 'G_W'.")
Else
        print(Argv(1) + " does not indicate 'G_W'.")
End If
```

Is_G_X - Check the File for the 'Group Execute' Attribute

Valid on UNIX only

The Is_G_X function checks the file for the specified attribute Group Execute.

Function format:

```
Is_G_X(filename as String) as Boolean
```

name

Identifies a file path.

This function returns TRUE if the attribute is set and FALSE otherwise.

Example:

```
If Is_G_X(Argv(1)) Then
        print(Argv(1) + " indicates 'G_X'.")
Else
        print(Argv(1) + " does not indicate 'G_X'.")
End If
```

Is_Hidden - Check the File for the 'Hidden' Attribute

Valid on Windows

The Is_Hidden function checks the file for the specified attribute.

Function format:

```
Is_Hidden(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the file is a hidden file; otherwise, returns FALSE.

Example:

```
If Is_Hidden(Argv(1)) Then
        print(Argv(1) + " indicates 'Hidden'.")
Else
        print(Argv(1) + " does not indicate 'Hidden'.")
End If
```

Is_O_R - Check the File for the 'Others Read' Attribute

Valid on UNIX only

The Is_O_R function checks the file for the specified attribute Others Read.

Function format:

```
Is_O_R(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_O_R(Argv(1)) Then
        print(Argv(1) + " indicates 'O_R'.")
Else
        print(Argv(1) + " does not indicate 'O_R'.")
End If
```

Is_O_W - Check the File for the 'Others Write' Attribute

Valid on UNIX only

The Is_O_W function checks the file for the specified attribute Others Write.

Function format:

```
Is_O_W(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_O_W(Argv(1)) Then
        print(Argv(1) + " indicates 'O_W'.")
Else
        print(Argv(1) + " does not indicate 'O_W'.")
End If
```

Is_O_X - Check the File for the 'Others Execute' Attribute

Valid on UNIX only

The Is_O_X function checks the file for the specified attribute Others Execute.

Function format:

```
Is_O_X(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_O_X(Argv(1)) Then
        print(Argv(1) + " indicates 'O_X'.")
Else
        print(Argv(1) + " does not indicate 'O_X'.")
End If
```

Is_ReadOnly - Check the File for the 'Read Only' Attribute

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Is_ReadOnly function checks the file for the specified attribute.

Function format:

```
Is_ReadOnly(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_ReadOnly(Argv(1)) Then
        print(Argv(1) + " indicates 'ReadOnly'.")
Else
        print(Argv(1) + " does not indicate 'ReadOnly'.")
End If
```

Is_SGID - Check the File for the 'Set Group ID' Attribute

Valid on UNIX only

The Is_SGID function checks the file for the specified attribute Set Group ID.

Function format:

```
Is_SGID(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_SGID(Argv(1)) Then
        print(Argv(1) + " indicates 'SGID'.")
Else
        print(Argv(1) + " does not indicate 'SGID'.")
End If
```

Is_SUID - Check the File for the 'SET USER ID' Attribute

Valid on UNIX only

The Is_SUID function checks the file for the specified attribute SET USER ID.

Function format:

```
Is_SUID(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_SUID(Argv(1)) Then
        print(Argv(1) + " indicates 'SUID'.")
Else
        print(Argv(1) + " does not indicate 'SUID'.")
End If
```

Is_System - Check the File for the 'Is System' Attribute

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Is_System function checks the file for the specified attribute IS SYSTEM file.

Function format:

```
Is_System(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_System(Argv(1)) Then
        print(Argv(1) + " indicates 'System'.")
Else
        print(Argv(1) + " does not indicate 'System'.")
End If
```

Is_U_R - Check the File for the 'User Read' Attribute

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Is_U_R function checks the file for the specified attribute User Read.

Function format:

```
Is_U_R(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_U_R(Argv(1)) Then
        print(Argv(1) + " indicates 'U_R'.")
Else
        print(Argv(1) + " does not indicate 'U_R'.")
End If
```

Is_U_W - Check the File for the 'User Write' Attribute

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Is_U_W function checks the file for the specified attribute USER WRITE.

Function format:

```
Is_U_W(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_U_W(Argv(1)) Then
        print(Argv(1) + " indicates 'U_W'.")
Else
        print(Argv(1) + " does not indicate 'U_W'.")
End If
```

Is_U_X - Check the File for the 'User Execute' Attribute

Valid on UNIX only

The Is_U_X function checks the file for the specified attribute USER eXecute.

Function format:

```
Is_U_X(filename as String) as Boolean
```

filename

Identifies a file path.

This function returns TRUE if the attribute is set; otherwise, returns FALSE.

Example:

```
If Is_U_X(Argv(1)) Then
        print(Argv(1) + " indicates 'U_X'.")
Else
        print(Argv(1) + " does not indicate 'U_X'.")
End If
```

MkDir - File System Function

MkDir is an alias for CreateDirectory function.

More information:

CreateDirectory or MkDir - Create a New Directory (see page 72)

RemoveDirectory or RmDir - Remove a Directory

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The RemoveDirectory or RmDir function removes the directory specified by pathstring.

Note: The specified directory must not be write-protected.

Function format:

RemoveDirectory(pathstring as String) as Boolean

RemoveDir(pathstring as String) as Boolean

pathstring

Identifies the directory that is to be removed. The specified directory:

- Must be empty.
- Must not be the current working directory.
- Must not be the root directory.

On successful completion, the function returns TRUE, otherwise returns FALSE.

Example:

```
'Create dir
CreateDirectory("C:\AMTest")

'Set active dir
SetDirectory("C:\AMTest")
MessageBox(" Current Directory is : " + GetDirectory(0) )

'Set Default to C:\
SetDirectory("C:\")

'Rename the directory
RenameDirectory("C:\AMTest","C:\AMTest_1")
```

```
If ExistDirectory("C:\AMTest_1") THEN
 RemoveDirectory("C:\AMTest_1")

Else
 MessageBox("Could not find directory : C:\AMTest_1")

Endif
```

RenameDirectory or RenDir - Rename or Move a Directory

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

Use the RenameDirectory or RenDir function to rename or move the specified directory.

Function format:

```
RenameDirectory(oldname as String, newname as String) as Boolean

RenDir(oldname as String, newname as String) as Boolean
```

oldname

Specifies the name of an existing directory.

newname

Identifies the new name of the directory.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
'Create dir
CreateDirectory("C:\AMTest")

'Set active dir
SetDirectory("C:\AMTest")
MessageBox(" Current Directory is : " + GetDirectory(0) )

'Set Default to C:\
SetDirectory("C:\")

'Rename the directory
RenameDirectory("C:\AMTest","C:\AMTest_1")
```

```
If ExistDirectory("C:\AMTest_1") THEN
 RemoveDirectory("C:\AMTest_1")

Else
 MessageBox("Could not find directory : C:\AMTest_1")
Endif
```

RenameFile - Rename or Move a File

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

Use the RenameFile or RenFile function to rename or move the specified file.

Function format:

```
RenFile(oldname as String, newname as String) as Boolean

RenameFile(oldname as String, newname as String) as Boolean
```

oldname

Specifies the name of an existing file.

newname

Identifies the new name of the file.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
HELP_GETFILEATTRIBUTES

If Not(ExistDirectory("c:\backup")) then CreateDirectory("c:\backup")
CopyFile("c:\autoexec.bat", "c:\backup\autoexec.bat")
RenameFile("c:\backup\autoexec.bat", "c:\backup\autoexec.bak")
```

SetFileAttributes - Set the File Attributes

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The SetFileAttributes function sets the attributes of the file or directory specified by filename.

Function format:

```
SetFileAttributes(filename as String,attr as Integer) as Boolean
```

filename

Specifies the file path.

attr

Indicates the value that is to be set.

Windows

Values of the file attributes on Windows:

READONLY

Value: 1

HIDDEN

Value: 2

SYSTEM

Value: 4

DIRECTORY

Value: 16

ARCHIVE

Value: 32

NORMAL

Value: 128

TEMPORARY

Value: 256

COMPRESSED

Value: 2048

OFFLINE

Value: 4096

Example:

This example reports the attributes on AUTOEXEC.BAT and switches the Hidden flag.

```
dim attr as integer
dim out as string

attr=GetFileAttributes("C:BAT")
if (attr<>-1) then
        out=out+"Attributes: "+chr(9)+"["
        if attr and FA_ARCHIVE then out=out+"A"
        if attr and FA_SYSTEM then out=out+"S"
        if attr and FA_HIDDEN then out=out+"H"
        if attr and FA_RDONLY then out=out+"R"

        out=out+"]"
        MsgBox("Information on C:\AUTOEXEC.BAT"+chr(10)+chr(10)+out)
else
        MsgBox("Cannot find C:\AUTOEXEC.BAT")
end if
if attr and FA_HIDDEN then
        attr = attr – FA_HIDDEN
        SetFileAttributes("C:\AUTOEXEC.BAT", attr)
else
        attr = attr + FA_HIDDEN
        SetFileAttributes("C:\AUTOEXEC.BAT", attr)
End If
end:
```

SetFileTime - Set the File or Directory Timestamps

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The SetFileTime function sets the timestamps of a file or directory.

Function format:

```
SetFileTime(fileName as string, dateAndTime as DateTime, Type as integer) as Boolean

SetFileTime(fileName as string, dateAndTime as DateTime) as Boolean
```

fileName

Specifies the name of the file for which the date and time is to be modified.

dateAndTime

Identifies the new date and time to be set.

Type

Value 0; Creation date and time are modified (default).

Value 1; Last access file date and time are modified.

Value 2; Last modified file date and time are modified.

Value 3; Creation, last access, and final write date and time are modified.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
Rem modify write time
Dim newTime as DateTime
newTime.year = 1999
newTime.month = 12
newTime.day = 31
newTime.hour = 23
newTime.minute = 59
newTime.second = 59
If Not(SetFileTime("c:\autoexec.bat",newTime,2)) Then
        Print ("SetFileTime failed")
        Exit
End If
```

Read and Compare File Versions

You can read and compare the version information of any file using DMScript functions. For example, you can read and compare the version of dmscript.exe to verify whether the script is running on the desired version of DMScript.

GetFileInfo

The GetFileInfo function reads the file information of a given file. You can use the retrieved file information in other functions such as CompareVersions.

This function has the following format:

```
GetFileInfo (option as string, filename as string, result as string) as boolean
```

Example: GetFileInfo

```
getfileinfo("version","dmscript.exe",version)
```

Input Parameters

This function has the following input parameters:

Option

Specifies the file information you want to retrieve. You can specify one of the following options:

"version"

Retrieves the complete file version string in the format major.minor.patch.build. Example: "12.5.0.1234"

"major"

Retrieves the major version number of the file.

"minor"

Retrieves the minor version number of the file.

"patch"

Retrieves the patch number of the file.

"build"

Retrieves the build number of the file.

"desc"

Retrieves the file description (Windows only).

"ext"

Retrieves the file extension.

"filename"

Retrieves the file name from a path name without the extension.

Result

Specifies a string variable that holds the result returned by the function.

Return Values

If the function is able to retrieve the file information, it returns true.

CompareVersions

The CompareVersions function compares two file versions and returns whether they are identical. This function has the following format:

```
CompareVersions (version1 as string, version2 as string)  as integer
```

Example: CompareVersions

```
CompareVersions ("12.5.1.4", "12.5.0.3")
```

Input Parameters

This function has the following input parameters:

version1 and version2

Specifies the versions of the file you want to compare. You can specify any number of version identification items in the version string. For example, you can specify the version string using the major.minor.patch.build format or just the major.minor format. The version numbers can be separated by dots, spaces, or commas. If one version string has more identification items than the other version string, the missing items are replaced with zero at the time of comparison. For example, CompareVersions ("12.5.180.123", "12.5") is treated as CompareVersions ("12.5.180.123", "12.5.0.0").

Return Values

The function returns one of the following predefined integer constants:

ECVEQUAL (0)

Indicates that the versions are equal.

ECVGREATERTHAN (1)

Indicates that version 1 is greater than version 2.

ECVLESSTHAN (-1)

Indicates that version 1 is lower than version 2.

ECVINVALID (2)

Indicates that one or both of the version strings is blank or contains a non-numeric value.

Icon Functions

AddItem - Add an Icon

Valid on Windows only

The AddItem function adds an icon to an existing group.

Function format:

```
AddItem(
        title as String,
        cmdline as String,
        iconfile as String,
        workdir as String,
        group as String,
        min as Boolean) as Boolean
```

title

Identifies the title to be displayed below the icon in the Group window.

cmdline

Specifies the full command line required to run the application. This parameter must be the name of the executable file for the application, but it can also include the full path of the application and any parameters that the application needs.

iconfile

Specifies the file name of the icon to be displayed in the Group window. This file can be an executable or an icon. If this parameter is an empty string, the first icon in the command-line executable is used.

workdir

Identifies the name of the default (or working) directory.

group

Identifies the program manager group that is to be the active group. If this parameter is an empty string, Windows uses the currently active group.

min

TRUE indicates that the executable is meant to starts in a minimized state.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

AddItem function returns a boolean value.

CreateGroup - Create a New Group or Activate an Existing Group

Valid on Windows only

The CreateGroup function creates a new group or activates the window of an existing group.

Function format:

```
CreateGroup (name as String) as Boolean

CreateGroup (name as String,
        offset as Integer) as Boolean
```

name

Identifies the group name.

offset

One of:

- LNK_PROGRAMS
- LNK_DESKTOP
- LNK_STARTMENU
- LNK_STARTUP
- LNK_ABSOLUTE

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

This example creates an icon group named DMS Test with an icon to the script editor. This example adds a shortcut on the desktop. The icon group and the shortcut are removed.

```
Rem
' determine location of dmsedit.exe
Dim hkey, dummy As Integer
Dim dmseditPath As String

hkey = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\ComputerAssociates\DMScript\DMSEdit")
If hKey = 0 Then
    MessageBox("Can not open DMSEdit-key", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(1)
    Exit
End If

If Not(RegQueryVariable(hkey, "dmseditInstalledAt", dmseditPath, dummy) = REG_STRING) Then
    MessageBox("dmseditInstalledAt not found or invalid", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(2)
    Exit
End If

RegCloseKey(hkey)

' Now Create group and icons
If Not(CreateGroup("DMS Test", LNK_PROGRAMS)) Then
    MessageBox("Can not create icon group", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(3)
    Exit
End If

If Not(AddItem("DMS Editor", dmseditPath+"\dmsedit.exe",'"','"', "DMS Test",FALSE,)) Then
    MessageBox("Failed to install dmsedit at icon group", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(4)
    Exit
End If

if Not(CreateLink("DMS Editor", dmseditPath+"\dmsedit.exe",'"','"', '"',LNK_NORMAL, LNK_DESKTOP)) Then
    MessageBox("Failed to install dmsedit at desktop", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(5)
    Exit
End If
```

```
MessageBox("Icons and shortcut has been created, press OK to delete them again","Desktop Management
Scripting: Icon Confirm", MB_OK + MB_ICONINFORMATION)

'Delete shortcut
if Not(DeleteItem("DMS Editor", "", LNK_DESKTOP)) Then
    MessageBox("Failed to deinstall dmsedit from desktop", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(6)
    Exit
End If

if Not(DeleteGroup("DMS Test", LNK_PROGRAMS)) Then
    MessageBox("Failed to delete icon group", "Desktop Management Scripting", MB_OK +
    MB_ICONEXCLAMATION)
    SetStatus(7)
    Exit
End If
```

CreateLink - Create a Shortcut

Valid on Windows only

The CreateLink function creates a shortcut to a file.

Default Format

CreateLink(Title as string, FileName as string, Arguments as string, WorkDir as string, Group as string, IconFile as string, Style as Integer, Offset as Integer) as Boolean.

Other Formats

CreateLink(Title as string, Filename as string, Arguments as string, Workdir as string, Group as string) as Boolean

as main format but with iconfile = "", style = LNK_NORMAL, and offset = LNK_PROGRAMS.

CreateLink(Title as string, Filename as string, Arguments as string, Workdir as string,Group as string, Iconfile as string) as Boolean

as the main format but with style = LNK_NORMAL and offset = LNK_PROGRAMS.

CreateLink(Title as string, Filename as string, Arguments as string, Workdir as string, Group as string, Style as integer) as Boolean

as the main format with iconfile = "" and offset = LNK_PROGRAMS.

CreateLink(Title as string, Filename as string, Arguments as string, Workdir as string, Group as string, Style as integer, Offset as integer) as Boolean

which is the same as the main format with iconfile = "".

CreateLink(Title as string, Filename as string, Arguments as string, Workdir as string, Group as string, Iconfile as string, Style as integer) as Boolean

as the main format but with offset = LNK_PROGRAMS. For software delivery style compatibility, TRUE is mapped to LNK_MINIMIZE and FALSE to LNK_NORMAL.

CreateLink(Title as string, Filename as string, Arguments as string, Workdir as string, Group as string, Iconfile as string, Iconindex as integer, Style as integer, Offset as integer, Hotkey as word, Description as string) as Boolean

Hotkey consists of two parts, the virtual-key code in the low-order byte and the modifier flag in the high-order byte. For the modifier flag, the following constants are defined:

HOTKEYF_ALT

Alt key

HOTKEYF_CONTROL

Ctrl key

HOTKEYF_EXT

Extended key

HOTKEYF_SHIFT

Shift key

To define a hotkey Ctrl+Alt+X you can code

```
Dim HOTKEYF as Word
hotKey = HHOTKEYF_ALT + HOTKEYF_CONTROL + asc("X")
```

Title

Specifies the title of the shortcut.

FileName

Specifies the path and name of the file to which the shortcut refers. As a minimum, this parameter must be the name of the file.

Arguments

Specifies optional arguments or switches to the file.

WorkDir

Specifies the name of the default (or working) directory.

Group

Identifies the group (or folder) to add the shortcut to. If this parameter is an empty string, the value of the Offset parameter determines where the shortcut is placed.

IconFile

Identifies the file name for the icon to be displayed in the group window. This file can be an executable file or an icon file. If this parameter is an empty string or omitted, the first icon in the file specified by FileName is used.

Style

Specifies how the file appears when the shortcut is activated. Use one of the following constants.

LNK_NORMAL

Runs as normal.

LNK_MAXIMIZE

Runs as maximized.

LNK_MINIMIZE

Runs as minimized. If Style is not specified, the value LNK_NORMAL is used.

Offset

Specifies the relative location of the *Group* argument. Use one of the following constants.

LNK_PROGRAMS

Places the item in the Programs group.

LNK_DESKTOP

Places the item on the Desktop.

LNK_STARTMENU

Places the item in the Start menu.

LNK_STARTUP

Places the item in the Startup group.

LNK_ABSOLUTE

Places the item in the specified directory.

Iconindex

Specifies the index of the icon to show. If iconindex is not coded, the first icon in the icon list is chosen.

Hotkey

Specifies hotkey code for the icon. If 0 is coded, no hotkey is set.

Description

Identifies transparent information stored in the shortcut.

On successful completion, the function returns TRUE; otherwise, returns FALSE. When running in the DOS environment, the command is placed in a queue to run on Windows startup. For this reason, the return value is always TRUE.

DeleteGroup - Delete a Group

Valid on Windows only

The DeleteGroup function removes the group specified by name.

Function format:

```
DeleteGroup(name as String) as Boolean
```

name

Identifies the name of the group to delete.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

DeleteItem - Remove an Item from a Group

Valid on Windows only

The DeleteItem function removes a specified item from the specified group.

Function format:

```
DeleteItem(name as String, group as String) as Boolean
```

name

Identifies the name of the item to be deleted.

group

Identifies the name of the group containing the item to be deleted.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

RemoveLink - Remove a Shortcut

Valid on Windows only

The RemoveLink function removes a shortcut.

Function format:

RemoveLink(Name as string, Group as string, Offset as integer) as Boolean

Name

Specifies the name of the shortcut to remove.

Group

Identifies the group (or folder) in which the shortcut to be removed is located. If this parameter is an empty string, the value of the *Offset* parameter determines where the shortcut will be removed.

LNK_PROGRAMS

The item is removed from the Programs group.

LNK_DESKTOP

The item is removed from the Desktop.

LNK_STARTMENU

The item is removed from the Start menu.

LNK_STARTUP

The item is removed from the Startup group.

LNK_ABSOLUTE

The item is removed from the specified directory.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
if Not(RemoveLink("DMS Editor", "", LNK_DESKTOP)) Then
        MessageBox("Failed to deinstall dmsedit from desktop", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)
        SetStatus(1)
        Exit
End If
```

Initialization File (.ini) Functions

You can use the Initialization (.ini) File functions on Windows NT, Windows 9x, and UNIX platforms.

Important! .ini files must not contain Japanese characters on Linux computers.

DeleteIniEntry - Remove an Entry in an Initialization File

Valid on Windows only

The DeleteIniEntry function removes a specified entry in a specified section of a specified initialization (.ini) file.

Function format:

```
DeleteIniEntry(section as String, entry as String, filename as String) as Boolean
```

section

Identifies the section that contains the entry.

entry

Identifies the entry to be deleted.

filename

Identifies the name of the initialization (.ini) file.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
Rem By deleting the file Ncclient.ini a new workstation name detect
Rem will be forced next time the AM agent is executed.
Rem
DeleteFile(ComputerPath+"Ncclient.ini")
Rem ComnputerPath is an Asset Management constant
```

DeleteIniSection - Remove a Section in an Initialization File

Valid on UNIX platforms

The DeleteIniSection function removes an entire section in the specified initialization (.ini) file.

Function format:

```
DeleteIniSection(section as String, filename as String) as Boolean
```

section

Identifies the section to delete.

filename

Indicates the name of the initialization (.ini) file.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

This example reads from the file c:\temp\test.ini, the section LIST and from the section SINGLE the entry Entry, and prints them. The example deletes the section LIST and the entry Entry from the section SINGLE.

```
Function strTok(in As string, BYREF pos As Integer, token As String, BYREF out As String) As Boolean
        Dim buf As String
        Dim len As Integer

        If (pos > Len(in)) Then
                strTok = FALSE
        Else
                buf = Mid(in, pos)
                len = Instr(buf, token)
                If len = 0 Then
                        out = buf
                        pos = pos + Len(buf) + 1
                Else
                        out = Mid(buf, 0, len-1)

pos = pos + len + 1
                End If
                strTok = TRUE
        End If
End Function

Dim attrList, str, lf As string
Dim pos As Integer

' clear screen
ClrScr()

If Not(ReadIniSection("LIST", attrList, "c:\temp\test.ini")) Then
        MessageBox("Can not read LIST section", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(1)
        Exit
End If
Print("[LIST]")
pos = 0
lf = chr(10)
While (strTok(attrList, pos, lf, str))
        Print(str)
Wend

If Not(ReadIniEntry("SINGLE", "Entry", attrList, "c:\temp\test.ini")) Then

MessageBox("Can not read SINGLE section", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(2)
        Exit
```

```
End If

Print("[SINGLE]" + NEWLINE$ + "Entry=" + attrList)

If Not(DeleteIniEntry("SINGLE", "Entry", "c:\temp\test.ini")) Then
        MessageBox("Can not delete [SINGLE].Entry ", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(3)
        Exit
End If
If Not(DeleteIniSection("LIST", "c:\temp\test.ini")) Then
        MessageBox("Can not delete LIST section", "DMS", MB_OK + MB_ICONEXCLAMATION)

SetStatus(4)
        Exit
End If
```

ReadIniSection - Retrieve a Section

Valid on UNIX and Windows platforms

The ReadIniSection function retrieves an entire section from an initialization (.ini) file.

Function format:

```
ReadIniSection( section as String, result as String, filename as String) as Integer
```

section

Identifies the section to be retrieved.

result

Identifies the string variable that is to receive the section.

filename

Identifies the name of the initialization (.ini) file. If only the file name is specified in a Windows environment, the Windows directory is searched, while in non-Windows environments, the current directory is searched.

ReadIniEntry() and ReadIniSection() return a positive value if the call succeeds; otherwise, they return zero (0). The function searches the initialization file for a section that matches the name specified by section and then copies all the entries to the result variable in the following format:

```
Entry_1 = Value_1 <LF>
Entry_n = Value_n <LF>
```

Example:

```
Function strTok(in As string, BYREF pos As Integer, token As String, BYREF out As String) As Boolean
        Dim buf As String
        Dim len As Integer

        If (pos > Len(in)) Then
                strTok = FALSE
        Else
                buf = Mid(in, pos)
                len = Instr(buf, token)
                If len = 0 Then
                        out = buf
                        pos = pos + Len(buf) + 1
                Else
                        out = Mid(buf, 0, len-1)

pos = pos + len + 1
                End If
                strTok = TRUE
        End If
End Function

Dim attrList, str, lf As string
Dim pos As Integer

' clear screen
ClrScr()

If Not(ReadIniSection("LIST", attrList, "c:\temp\test.ini")) Then
        MessageBox("Can not read LIST section", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(1)
        Exit
End If
Print("[LIST]")
pos = 0
lf = chr(10)
While (strTok(attrList, pos, lf, str))
        Print(str)
Wend

If Not(ReadIniEntry("SINGLE", "Entry", attrList, "c:\temp\test.ini")) Then

MessageBox("Can not read SINGLE section", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(2)
        Exit
End If

Print("[SINGLE]" + NEWLINE$ + "Entry=" + attrList)
```

```
If Not(DeleteIniEntry("SINGLE", "Entry", "c:\temp\test.ini")) Then
        MessageBox("Can not delete [SINGLE].Entry ", "DMS", MB_OK + MB_ICONEXCLAMATION)
        SetStatus(3)
        Exit
End If
If Not(DeleteIniSection("LIST", "c:\temp\test.ini")) Then
        MessageBox("Can not delete LIST section", "DMS", MB_OK + MB_ICONEXCLAMATION)

SetStatus(4)
        Exit
End If
```

WriteIniEntry - Store a Value in an .ini file

Valid on UNIX and Windows

The WriteIniEntry function stores a value in an initialization (.ini) file. If the file does not exist, this function creates the file.

Function format:

WriteIniEntry(section as String, entry as String, value as String, grpfilename as String) as Boolean

WriteIniEntry(filename as String) as Boolean (Windows only)

section

Identifies the section in which to store the entry and value.

entry

Specifies the entry in which to store the value.

value

Identifies the value to be stored.

grpfilename

Identifies the name of the initialization file.

filename

The name of the addressed initialization file.

Note: In the case of WriteIniEntry{Filename as String} as Boolean, the cache of the system is flushed and therefore the new values become active without reboot.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
Dim file, section, entry, value as string

Dim LF, CR as char
Dim rtr as integer

ClrScr()
LF = 0x0a
CR = 0x0d
file = "c:\dmscript.ini"
if Not(ExistFile(file)) then
        rtr = CreateFile(file, O_TEXT)
        if rtr = -1 then
                SetStatus(1)
                exit
        end if
        closeFile(rtr)
end if

section = "Section 1"
value = "Param_1 = Wert_1" + LF + "Param_2 = Wert_2"

if WriteIniSection( section, value, file) then
        Print("WriteIniSection successfully completed.")
else
        Print("WriteIniSection failed.")
endif

section = "Section 2"
entry = "Param_3"
value = "Wert_3"

if WriteIniEntry( section, entry, value, file) then
        Print("WriteIniFile successfully completed.")
else
        Print("WriteIniFile failed.")
endif

section = "Section 1"
rtr = ReadIniSection(section, value, file)
if (rtr > 0) then
        Print( Str(rtr) + CR + LF + value)
else
        Print("ReadIniSection failed.")
endif
```

```
section = "Section 2"
entry = "Param_3"
rtr = ReadIniEntry(section, entry, value, file)

if (rtr > 0) then
        Print( Str(rtr) + CR + LF + entry + " = " + value)
else
        Print("ReadIniEntry failed.")

endif
```

WriteIniSection - Create or Overwrite an Entire Section

Valid on UNIX and Windows

The WriteIniSection function creates or overwrites an entire section in a specific initialization (.ini) file. If the .ini file does not exist, this function creates it.

Function format:

```
WriteIniSection(section as String, value as String, filename as String) as Boolean
```

section

Identifies the name of the section to be written.

value

Identifies the entries to be written. The entries in value must have the following format:

```
Entry_1 = Value_1 <LF>

Entry_n = Value_n <LF>
```

filename

Specifies the name of the initialization file.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
Dim entries as String
Dim LF as String
LF = chr(10)
entries = "Entry_1 = Value_1" + LF + "Entry_n = Value_n"
Rem         this example writes to c:\temp\test.ini the section LIST

Dim attrList as String

lf = chr(10)
attrList = "Entry_1=Value_1"+lf+"Entry_2=Value_2"

If Not(WriteIniSection("LIST", attrList, "c:\temp\test.ini")) Then
            MessageBox("Can not create LIST section", "DMS", MB_OK + MB_ICONEXCLAMATION)
            SetStatus(1)
            Exit
End If
```

ReadIniEntry - Retrieve a Value from a Specified Section

Valid on UNIX and Windows

The ReadIniEntry function retrieves a value from a specified section in an initialization (.ini) file.

Function format:

```
ReadIniEntry(section as String, entry as String, result as String, filename as String) as Integer
```

section

Identifies the section containing the entry.

entry

Identifies the entry whose associated string is to be retrieved.

filename

Identifies the name of the initialization (.ini) file. If only the file name is specified in Windows environments, the Windows directory is searched, while in non-Windows environments, the current directory is searched.

result

Identifies the string variable that is to receive the value.

ReadIniEntry() and ReadIniSection() return an positive value if the call succeeds, otherwsise they return zero (0). This function searches the initialization file for an entry that matches the name specified by entry, under the section heading specified by section and then copies its associated string to the result variable.

Example:

```
Dim file, section, entry, value as string

Dim LF, CR as char
Dim rtr as integer

ClrScr()
LF = 0x0a
CR = 0x0d
file = "c:\dmscript.ini"
if Not(ExistFile(file)) then
        rtr = CreateFile(file, O_TEXT)
        if rtr = -1 then
                SetStatus(1)
                exit
        end if
        closeFile(rtr)
end if

section = "Section 1"
value = "Param_1 = Wert_1" + LF + "Param_2 = Wert_2"

if WriteIniSection( section, value, file) then
        Print("WriteIniSection successfully completed.")
else
        Print("WriteIniSection failed.")
endif

section = "Section 2"
entry = "Param_3"
value = "Wert_3"

if WriteIniEntry( section, entry, value, file) then
        Print("WriteIniFile successfully completed.")
else
        Print("WriteIniFile failed.")
endif

section = "Section 1"
rtr = ReadIniSection(section, value, file)
if (rtr > 0) then
        Print( Str(rtr) + CR + LF + value)
else
        Print("ReadIniSection failed.")
endif
```

```
section = "Section 2"
entry = "Param_3"
rtr = ReadIniEntry(section, entry, value, file)

if (rtr > 0) then
        Print( Str(rtr) + CR + LF + entry + " = " + value)
else
        Print("ReadIniEntry failed.")

endif
```

MIF and Inv File Functions

CreateMIFFile - Create an MIF File

Valid on UNIX and Windows

CreateMIFFile function creates an MIF file.

Function format:

```
CreateMIFFile(Filename as string, Name as string, Description as string) as integer
```

Filename

Specifies the file name for the MIF file.

Name

Specifies the Component Name to store in the MIF file.

Description

Specifies the Description to store in the MIF file.

The return value of the function is an integer.

Example:

This example stores the Current Directory on C: into a MIF file, together with the free disk space. This is stored in the Workstation Directory. The example works only if asset management is installed.

```
Dim Dir as String
Dim DriveFree as Integer

'Get directory
Dir = GetDirectory(3)
DriveFree = GetDiskFreeMB(3)

'Create .MIF file
CreateMIFFile(ComputerPath+"Disk.MIF","My Disk Inventory ","","")
CreateMIFGroup(ComputerPath+"Disk.MIF","Disk Info","Disk information","AMClass")
CreateMIFString(ComputerPath+"Disk.MIF","Disk Info","Current Dir",Dir,"Dir name")
CreateMIFInteger(ComputerPath+"Disk.MIF","Disk Info","Free in MB",DriveFree,"Free MB")
```

CreateMIFGroup - Create a Group

Valid on UNIX and Windows

CreateMIFGroup creates a group in an existing MIF file.

Function format:

```
CreateMIFGroup(Filename as string, Name as string, Description as string, Class as string) as integer
```

Filename

Specifies the file name of the MIF file in which to create the group.

Name

Specifies the Name of the group to create.

Description

Specifies the Description of the group.

Class

Specifies the Class of the group.

On successful completion, the function returns the ID assigned to the group in the MIF file; otherwise, returns zero.

Example:

```
Dim Dir as String
Dim DriveFree as Integer

'Get directory
Dir = GetDirectory(3)
DriveFree = GetDiskFreeMB(3)

'Create .MIF file
CreateMIFFile(ComputerPath+"Disk.MIF","My Disk Inventory ","","")
CreateMIFGroup(ComputerPath+"Disk.MIF","Disk Info","Disk information","AMClass")
CreateMIFString(ComputerPath+"Disk.MIF","Disk Info","Current Dir",Dir,"Dir name")
CreateMIFInteger(ComputerPath+"Disk.MIF","Disk Info","Free in MB",DriveFree,"Free MB")
```

CreateMIFInteger - Create an Integer Attribute

Valid on UNIX and Windows

This function creates an integer attribute in an existing group in an existing MIF file.

Function format:

CreateMIFInteger(*Filename* as string, *Group* as string, *Name* as string, *Value* as integer, *Description* as string) as integer

CreateMIFInteger(*Filename* as string, *Group* as integer, *Name* as string, *Value* as integer, *Description* as string) as integer

Filename

Specifies the file name of the MIF file in which to create the Integer attribute.

Group

Specifies the group in which the Integer attribute is created. It can be a string specifying the name of the group or an integer specifying the ID of the group. The ID is the value returned by a call to the CreateMIFGroup function.

Name

Specifies the name of the attribute.

Value

Integer specifying the value of the attribute.

Description

Specifies the description of the attribute.

On successful completion, the function returns the ID of the attribute within the group in the MIF file; otherwise, returns zero.

Example:

```
Dim Dir as String
Dim DriveFree as Integer

'Get directory
Dir = GetDirectory(3)
DriveFree = GetDiskFreeMB(3)

'Create .MIF file
CreateMIFFile(ComputerPath+"Disk.MIF","My Disk Inventory ","","")
CreateMIFGroup(ComputerPath+"Disk.MIF","Disk Info","Disk information","AMClass")
CreateMIFString(ComputerPath+"Disk.MIF","Disk Info","Current Dir",Dir,"Dir name")
CreateMIFInteger(ComputerPath+"Disk.MIF","Disk Info","Free in MB",DriveFree,"Free MB")
```

CreateMIFString - Create a String Attribute

Valid on UNIX and Windows

CreateMIFString creates a string attribute in an existing group in an existing MIF file.

Function format:

```
CreateMIFString (Filename as String, Group as String, Name as String, Value as String, Description as String) as Integer
```

Filename

Specifies the name of the MIF file in which to create the string attribute.

Group

Specifies the group in which the string attribute is created. It can be a string specifying the name of the group or an integer specifying the ID of the group. The ID is the value returned by a call to the CreateMIFGroup function.

Name

Specifies the name of the attribute.

Value

String specifying the value of the attribute.

Description

Specifies the description of the attribute.

On successful completion, the function returns the ID of the attribute within the group in the MIF file; otherwise, returns zero.

Example:

```
Dim Dir as String
Dim DriveFree as Integer

'Get directory
Dir = GetDirectory(3)
DriveFree = GetDiskFreeMB(3)

'Create .MIF file
CreateMIFFile(ComputerPath+"Disk.MIF","My Disk Inventory "," "," ")
CreateMIFGroup(ComputerPath+"Disk.MIF","Disk Info","Disk information","AMClass")
CreateMIFString(ComputerPath+"Disk.MIF","Disk Info","Current Dir",Dir,"Dir name")
CreateMIFInteger(ComputerPath+"Disk.MIF","Disk Info","Free in MB",DriveFree,"Free MB")
```

GetMIFComponent or GetComponent - Retrieve the Component Name

Valid on UNIX and Windows

GetMIFComponent or GetComponent retrieves the component name of an existing .MIF or Inv file.

Function format:

```
GetMIFComponent(Filename as string) as string

GetComponent(Filename as string) as string
```

Filename

Specifies the file name of the MIF file to retrieve the component from.

On successful completion, the function returns a string containing the component name; otherwise, returns an empty string.

Example:

The following is an example of the GetMIFComponent, GetMIFInteger, GetMIFString, GetMIFValue, and SetMIFValue functions.

```
Dim file, gname, aname as string
Dim gid as integer

ClrScr()
file = "h:\test\miffus\file.mif"

Print("Component of '""" + file + """: '""" + GetMIFComponent(file) + """")

gname = "Strings"
gId = 1
aname = "string_1"
Print(gname + "." + aname + " = " + GetMIFString(file, gname, gId, aname, 1))
aname = "string_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))

if SetMIFValue(file, gname, gId, aname, 2, "Text_new") then
        Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("1. SetMIFValue failed.")
        exit
endif

gname = "Numbers"
gId = 2
aname = "number_1"
Print(gname + "." + aname + " = " + Str(GetMIFInteger(file, gname, gId, aname, 1)))
aname = "number_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
if SetMIFValue(file, gname, gId, aname, 2, "999999") then

Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("2. SetMIFValue failed.")
        exit
endif
```

GetMIFInteger - Retrieve the Value of an Integer Attribute

Valid on UNIX and Windows

GetMIFInteger retrieves the value of a specific Integer attribute from a MIF file.

Function format:

GetMIFInteger(Filename as string, GroupName as string, AttrName as string) as integer

GetMIFInteger(Filename as string, GroupID as integer, AttrID as integer) as integer

GetMIFInteger(Filename as string, GroupName as string, GroupID as integer, AttrName as string, AttrID as integer) as integer

Filename

File name of the MIF file.

GroupName

Specifies the name of the group in which the attribute is located.

GroupID

Specifies the ID of the group in which the attribute is located.

AttrName

Specifies the name of the attribute to retrieve the value from.

AttrID

Specifies the ID of the attribute to retrieve the value from.

If the attribute specified is found in the MIF file, GetMIFInteger returns the value of the Integer attribute; otherwise, 0 is returned.

Example:

The following is an example of the GetMIFComponent, GetMIFInteger, GetMIFString, GetMIFValue, and SetMIFValue functions.

```
Dim file, gname, aname as string
Dim gid as integer

ClrScr()
file = "h:\test\miffus\file.mif"

Print("Component of """ + file + """: """ + GetMIFComponent(file) + """")
```

```
gname = "Strings"
gId = 1
aname = "string_1"
Print(gname + "." + aname + " = " + GetMIFString(file, gname, gId, aname, 1))
aname = "string_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))

if SetMIFValue(file, gname, gId, aname, 2, "Text_new") then
        Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("1. SetMIFValue failed.")
        exit
endif

gname = "Numbers"
gId = 2
aname = "number_1"
Print(gname + "." + aname + " = " + Str(GetMIFInteger(file, gname, gId, aname, 1)))
aname = "number_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
if SetMIFValue(file, gname, gId, aname, 2, "999999") then

Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("2. SetMIFValue failed.")
        exit
endif
```

GetMIFString - Retrieve the Value of a String Attribute

Valid on UNIX and Windows

The GetMIFString function retrieves the value of a specific string attribute from a MIF file.

Function format:

GetMIFString(Filename as string, GroupName as string, AttrName as string) as string

GetMIFString(Filename as string, GroupID as integer, AttrID as integer) as string

GetMIFString(Filename as string, GroupName as string, GroupID as integer, AttrName as string, AttrID as integer) as string

Filename

Specifies the name of the MIF file.

GroupName

Specifies the name of the group in which the attribute is located.

GroupID

Specifies the ID of the group in which the attribute is located.

AttrName

Specifies the name of the attribute to retrieve the value from.

AttrID

Specifies the ID of the attribute to retrieve the value from.

If the attribute specified is found in the MIF file, GetMIFString returns a string representing the value of the attribute; otherwise, returns an empty string.

Example:

The following is an example of the GetMIFComponent, GetMIFInteger, GetMIFString, GetMIFValue, and SetMIFValue functions.

```
Dim file, gname, aname as string
Dim gid as integer

ClrScr()
file = "h:\test\miffus\file.mif"

Print("Component of '" + file + "': '" + GetMIFComponent(file) + "'")

gname = "Strings"
gId = 1
aname = "string_1"
Print(gname + "." + aname + " = " + GetMIFString(file, gname, gId, aname, 1))
aname = "string_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))

if SetMIFValue(file, gname, gId, aname, 2, "Text_new") then
        Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("1. SetMIFValue failed.")
        exit
endif
```

```
gname = "Numbers"
gId = 2
aname = "number_1"
Print(gname + "." + aname + " = " + Str(GetMIFInteger(file, gname, gId, aname, 1)))
aname = "number_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
if SetMIFValue(file, gname, gId, aname, 2, "999999") then

Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("2. SetMIFValue failed.")
        exit
endif
```

GetMIFValue - Retrieve the Value of an Attribute

Valid on UNIX and Windows

The GetMIFValue function retrieves the value of a specific attribute from a MIF file.

Function format:

GetMIFValue(Filename as string, GroupName as string, AttrName as string) as string

GetMIFValue(Filename as string, GroupID as integer, AttrID as integer) as string

GetMIFValue(Filename as string, GroupName as string, GroupID as integer, AttrName as string, AttrID as integer) as string

Filename

Identifies the file name of the MIF file.

GroupName

Specifies the name of the group in which the attribute is located.

GroupID

Specifies the ID of the group in which the attribute is located.

AttrName

Specifies the name of the attribute to retrieve the value from.

AttrID

Specifies the ID of the attribute to retrieve the value from. If the value is a text string, the returned string includes a quotation mark before and after the string.

If the attribute specified is found in the MIF file, GetMIFValue returns a string representing the value of the attribute; otherwise, returns an empty string.

Example:

The following is an example of the GetMIFComponent, GetMIFInteger, GetMIFString, GetMIFValue, and SetMIFValue functions.

```
Dim file, gname, aname as string
Dim gid as integer

ClrScr()
file = "h:\test\miffus\file.mif"

Print("Component of '" + file + "': " + GetMIFComponent(file) + "'")

gname = "Strings"
gId = 1
aname = "string_1"
Print(gname + "." + aname + " = " + GetMIFString(file, gname, gId, aname, 1))
aname = "string_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))

if SetMIFValue(file, gname, gId, aname, 2, "Text_new") then
        Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("1. SetMIFValue failed.")
        exit
endif

gname = "Numbers"
gId = 2
aname = "number_1"
Print(gname + "." + aname + " = " + Str(GetMIFInteger(file, gname, gId, aname, 1)))
aname = "number_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
if SetMIFValue(file, gname, gId, aname, 2, "999999") then

Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("2. SetMIFValue failed.")
        exit

endif
```

SetComment - Copy the Text to UAM Job Comment Field

Valid on UNIX and Windows

Copies the string specified by the parameter *Text* to the Asset Management Job Comment field.

Function format:

SetComment(Text as string) as integer

text

Specifies the comment text.

On successful completion, the function returns TRUE, otherwise it returns FALSE.

Example:

This script shows the use of the SetStatus function.

```
if MsgBox("Should this job return an error?",MB_YESNO)=IDYES then
        SetStatus(NC_ERROR)
else
        SetStatus(NC_OK)
        Print ("The user selected OK when prompted...")
end if
```

SetInvValue - Change the Attribute Value

Valid on UNIX and Windows

The SetInvValue function changes the value of a specific attribute in an Inv file.

Function format:

SetInvValue(Filename as string, GroupName as string, GroupID as string, AttrName as string, AttrID as string, Value as string) as string

Filename

Specifies the filename of the Inv file.

GroupName

Specifies the name of the group in which the attribute is located.

GroupID

Specifies the ID of the group in which the attribute is located.

AttrName

Specifies the name of the attribute to be set.

AttrID

Specifies the ID of the attribute to be set.

Value

Specifies the value to be set to the attribute in the Inv file.

For Inv structured files, the GroupName specifies the complete group name and GroupID the related index number including type; the AttrName is the name of the attribute and AttrID specifies the related index including type and display type.

If the attribute specified is found in the Inv file, its value is changed and SetInvValue returns TRUE; otherwise, returns FALSE.

Example:

```
SetInvValue("sample.inc","$GeneralInventory$|$Drive$,"000A02","name","00010500","C:")
```

SetLocalPath - Copy the String to the UAM Job Local Path Field

Valid on UNIX and Windows

The SetLocalPath function copies the string specified by the parameter Text to the Asset Management (UAM) Job Local Path field.

Function format:

```
SetLocalPath(Text as string) as Boolean
```

Text

Defines the local path, for example, C:\TEMP\NEW.

On successful completion, the function returns TRUE (nonzero); otherwise, returns FALSE (zero).

Example:

```
If (SetLocalPath("C:\TEMP\NEW"))
Then Print ("SetLocalPath succeeded")
Else
        Print ("SetLocalPath failed")
Endif
```

SetMIFValue - Change the Attribute Value

Valid on UNIX and Windows

The SetMIFValue function changes the value of a specific attribute in a MIF file.

Function format:

SetMIFValue(Filename as string, GroupName as string, AttrName as string, Value as string) as Boolean

SetMIFValue(Filename as string, GroupID as integer, AttrID as integer, Value as string) as Boolean

SetMIFValue(Filename as string, GroupName as string, GroupID as integer, AttrName as string, AttrID as integer, Value as string) as Boolean

Filename

Specifies the filename of the MIF file

GroupName

Specifies the name of the group in which the attribute is located.

GroupID

Specifies the ID of the group in which the attribute is located.

AttrName

Specifies the name of the attribute to be set.

AttrID

Specifies the ID of the attribute to be set.

Value

Specifies the value to set the attribute to in the MIF file.

To assign a new integer to an integer attribute code:

- value="123456"

To assign a new string, the string must be enclosed in additional quotes; otherwise, a subsequent MIF function may produce unexpected results:

- value="""123456"""

If the attribute specified is found in the MIF file, its value is changed and SetMIFValue returns TRUE; otherwise, returns FALSE.

Example:

The following is an example of the GetMIFComponent, GetMIFInteger, GetMIFString, GetMIFValue, and SetMIFValue functions.

```
Dim file, gname, aname as string
Dim gid as integer

ClrScr()
file = "h:\test\miffus\file.mif"

Print("Component of '" + file + "': '" + GetMIFComponent(file) + "'")

gname = "Strings"
gId = 1
aname = "string_1"
Print(gname + "." + aname + " = " + GetMIFString(file, gname, gId, aname, 1))
aname = "string_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))

if SetMIFValue(file, gname, gId, aname, 2, "Text_new") then
        Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("1. SetMIFValue failed.")
        exit
endif

gname = "Numbers"
gId = 2
aname = "number_1"
Print(gname + "." + aname + " = " + Str(GetMIFInteger(file, gname, gId, aname, 1)))
aname = "number_2"
Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
if SetMIFValue(file, gname, gId, aname, 2, "999999") then

Print(gname + "." + aname + " = " + GetMIFValue(file, gname, gId, aname, 2))
else
        Print("2. SetMIFValue failed.")
        exit

endif
```

Miscellaneous Functions

Argc - Return the Number of Arguments Passed to the Script

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The argc function returns the number of arguments passed to the script. At least one argument, the name of the script, is always present.

Function format:

```
Argc() as Integer
```

The function returns an integer, which is the number of arguments passed to the script. A value of 1 indicates that only the script was passed.

Example:

```
Dim i as Integer
For i = 0 To argc() - 1
        Print(argv(i))
Next i
```

Argv - Returns a String Containing the nth Script Argument

Valid on UNIX and Windows

The Argv function returns a string containing the *n*th script argument. argv(0) is the name of the script.

Function format:

```
Argv(n as Integer) as String
```

n

Index to argument

The return value is a string that contains the *n*th script argument.

Example:

```
Dim i as Integer
For i = 0 To argc() - 1
        Print(argv(i))
Next i
```

ChDir or SetDirectory - Change or Set a Directory

Valid on UNIX and Windows

The ChDir or SetDirectory function changes the current directory.

Function format:

ChDir(pathstring as String) as Boolean

SetDirectory(pathstring as String) as Boolean

pathstring

Name of the directory.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
'Create dir
CreateDirectory("C:\AMTest")

'Set active dir
SetDirectory("C:\AMTest")
MessageBox(" Current Directory is : " + GetDirectory(0) )

'Set Default to C:\
SetDirectory("C:\")

'Rename the directory
RenameDirectory("C:\AMTest","C:\AMTest_1")

If ExistDirectory("C:\AMTest_1") THEN
 RemoveDirectory("C:\AMTest_1")

Else
 MessageBox("Could not find directory : C:\AMTest_1")

Endif
```

Delay or Sleep - Wait Before Script Execution

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Delay or Sleep function waits for n milliseconds before continuing with the script execution.

Function format:

Delay(n as Integer) as Boolean

Sleep(n as Integer) as Boolean

n

Specifies the delay time in milliseconds. One second = delay(1000).

The function always returns TRUE.

Examples:

delay(50)

Wait for 50 milliseconds for script execution.

delay(1000)

Wait for 1 second for script execution.

Exec or Execute - Execute an Application

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Exec or Execute function runs the specified application.

Function format:

Execute(cmdline as String, wait as Boolean, style as Integer) as Integer

Execute(cmdline as String, wait as Boolean) as Integer

Execute(cmdline as String) as Integer

cmdline

Identifies the command line (filename plus optional parameters).

Note: On Windows platforms, if the file to be executed does not have the extension .exe, you need to explicitly specify the file extension; for example, on Win9x, specify "command.com" instead of "command".

wait

Specifies whether the script waits until the application terminates or not. (Windows only)

style

Specifies the window style of the application. (Windows only)

Note: If a command in the command line contains a space, enclose the command in quotation marks. For example: Execute("""\temp 1\prgedit.exe""",true,0).

The wait parameter specifies if the function returns to script execution without waiting for the application to terminate. The value TRUE forces the function to wait for the application to terminate. The default is TRUE.

On a UNIX platform, the wait and the style parameters are ignored. The function always waits for the application to terminate before returning to script execution.

The style parameter specifies how to display a Windows based application main window. This parameter can be any of the following values:

Value: 0

Hides the window and passes activation to another window.

Value: 1

Activates a window and displays it in its default size and position. This is the default.

Value: 2

Activates a window and displays it as an icon.

Value: 3

Activates a window and displays it as a maximized window.

Value: 7

Displays a window as an icon. The window currently active remains active.

On Windows, if the function succeeds, a non-negative return code indicates the return code of the specified command invoked by the execute() function. If the interpreter fails to start the command, the following return codes are returned to indicate the problem:

Return Code: -1

The system is out of memory or out of resources.

Return Code: -2

The specified file was not found.

Return Code: -3

The specified path was not found.

Return Code: -4

The .exe file is invalid.

Return Code: -5

Unknown error.

On UNIX, the system function "system()" invokes the specified command. Therefore, the returned code corresponds to the return code of "system()". In other words, when the system() fails, it returns -1. When system() is successful, the return value corresponds to the "waitpid()" return code format. A script must analyze the return code to determine the success or failure of the invoked commands. This can be achieved by using the following functions:

```
'
' Execute() return code evaluation support for Unix
'

' ****************************************************************************************************
'
' Name
'   DMS_ExecExit(retVal As Integer) As Boolean
'
' Description
'   Determines whether the process started by Execute() has terminated normally or not.
'
' Arguments
'   retVal      return code to be evaluated
'
' Returns
'   TRUE    if normally terminated
'   FALSE   else.
'
' Comment
'  The function corresponds to the waitpid(5) description of the macro WIFEXITED
'               #define WIFEXITED(stat)     ((int)((stat)&0xFF) == 0)
'
Function DMS_ExecExit(retVal As Integer) As Boolean
                If (retVal And 255) Then
                                DMS_ExecExit = FALSE
                Else
                                DMS_ExecExit = TRUE
                End If
End Function
```

```
' ********************************************************************************************
'
' Name
'   DMS_ExecExitStatus(retVal As Integer) As Integer
'
' Description
'   Determines the shell exit code from the return value of the Execute() call, i.e. the return code of
'   the invoked process.
'
' Arguments
'   retVal      Retrun code retuned from Execute()
'
' Returns
'   If the Execute() has terminated normally the reported status of the child is returned, otherwise
'   retVal is returned.
'
' Comment
'   The function corresponds to the waitpid(5) description of the macro WEXITSTATUS
'               #define WEXITSTATUS(stat)   ((int)(((stat)>>8)&0xFF))
'
Function DMS_ExecExitStatus(retVal As Integer) As Integer
        If Not(DMS_ExecExit(retVal)) Then
                DMS_ExecExitStatus = retVal
        Else
                DMS_ExecExitStatus = retVal / 256
                DMS_ExecExitStatus = DMS_ExecExitStatus And 255
        End If
End Function
```

If the function succeeds, the return value is non-negative (the exit code of the application when WAIT specifies that the function should wait for the application to terminate). Otherwise, the function returns one of the following error values:

Value: -1

The system is out of memory or resources.

Value: -2

The specified file was not found.

Value: -3

The specified path was not found.

Value: -4

The .exe file is invalid.

Value: -5

Unknown error.

Example:

```
Dim Tempname as string  ' Holds filename of temporary file
Dim hFile as integer   ' Handle of temporary file

' First obtain temporary filename and create file...
Tempname=EnvGetString("TEMP")
if TempName="" then
        TempName="MyFile.txt"
else
        Tempname=Tempname+"\MyFile.txt"
end if
hfile=CreateFile(Tempname)

if hFile<0 then
        ' An error occured while creating file, inform user and exit.
        MsgBox("Could not create temporary file...")
        goto end
end if

' Store some text in the file and close it...
WriteFile(hFile,"This text is shown using notepad")
WriteFile(hFile,"")
WriteFile(hFile,"Please exit notepad to resume script execution")
Closefile(hFile)

'Launch notepad with the file...
if Execute("notepad "+Tempname,TRUE)<0 then MsgBox("Could not launch notepad...")

'remember to Erase the temporary file before leaving...
DeleteFile(Tempname)

end:
```

Exit - Quit the Script

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Exit function quits the script.

Function format:

```
exit()
```

The function does not return a value.

Not - Return the Logical Negation of an Expression

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Not function returns the logical negation of an expression.

Function format:

Not(n as Integer) as Integer

n

Expression.

If the parameter *n* evaluates to zero, the function returns TRUE (nonzero). Otherwise, returns FALSE (zero).

Example:

```
rem

Rem This example creates an icon group called DMS Test with an icon to the script editor.
Rem This Example also adds a shortcut on the desktop
Rem
Rem After the creation the icon group and the shortcut will be removed
Rem

' determine location of dmsedit.exe
Dim hkey, dummy As Integer
Dim dmseditPath As String

hkey = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\ComputerAssociates\DMScript\DMSEdit")
If hKey = 0 Then
        MessageBox("Can not open DMSEdit-key", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)

SetStatus(1)
        Exit
End If

If Not(RegQueryVariable(hkey, "dmseditInstalledAt", dmseditPath, dummy) = REG_STRING) Then
        MessageBox("dmseditInstalledAt not found or invalid", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)
        SetStatus(2)
        Exit
End If
```

```
RegCloseKey(hkey)

' Now Create group and icons
If Not(CreateGroup("DMS Test", LNK_PROGRAMS)) Then
        MessageBox("Can not create icon group", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)

SetStatus(3)
        Exit
End If

If Not(AddItem("DMS Editor", dmseditPath+"\dmsedit.exe",'"','"', "DMS Test",FALSE,)) Then
        MessageBox("Failed to install dmsedit at icon group", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)
        SetStatus(4)
        Exit
End If

if Not(CreateLink("DMS Editor", dmseditPath+"\dmsedit.exe",'"','"', '"',LNK_NORMAL, LNK_DESKTOP)) Then
        MessageBox("Failed to install dmsedit at desktop", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)

SetStatus(5)
        Exit
End If

MessageBox("Icons and shortcut has been created, press OK to delete them again","Desktop Management
Scripting: Icon Confirm", MB_OK + MB_ICONINFORMATION)

'Delete shortcut
if Not(DeleteItem("DMS Editor", '"', LNK_DESKTOP)) Then
        MessageBox("Failed to deinstall dmsedit from desktop", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)
        SetStatus(6)
        Exit
End If

if Not(DeleteGroup("DMS Test", LNK_PROGRAMS)) Then
        MessageBox("Failed to delete icon group", "Desktop Management Scripting", MB_OK +
MB_ICONEXCLAMATION)

SetStatus(7)
        Exit

End If
```

Print - Output the String Text to the Screen

Valid on UNIX and Windows

The Print function outputs the string text to the screen.

Function format:

```
Print(text as String) as Boolean
```

text

Text to be printed.

The function always returns TRUE.

Example:

```
Dim counter as integer

'Print 5 lines
For counter = 1 to 5
 Print("Line " + str(counter))
Next

'Wait 5 seconds
Sleep(5000)

'Clear the print screen
ClearScreen()
Print("Screen Cleared")
```

SetDirectory - Change the Current Directory

Valid on Windows only

The SetDirectory function changes the current directory.

Function format:

```
SetDirectory(pathstring as String) as Boolean
```

pathstring

Specifies the name of the directory to create.

On successful completion, the function returns TRUE, otherwise returns FALSE.

Example:

```
Rem switch to c:\backup
If Not(SetDirectory(c:\backup) Then
        Print("c:\backup does not exist");
        exit
End If
```

SetDrive - Change the Current Drive

Valid on Windows only

The SetDrive function changes the current drive.

Function format:

```
SetDrive(n1 as Integer) as Boolean
```

n1

Indicates the drive number. The parameter n1 specifies the drive as follows: 1 for A, 2 for B, 3 for C, and so on.

On successful completion, the function returns TRUE, otherwise, returns FALSE.

Example:

```
Rem switch to drive d.
If Not(SetDrive(4)) Then
        Print("Drive 4 not available");
        exit
End If
```

SetStatus - Return Status Value

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The SetStatus function returns a status value from the script.

Function format:

```
SetStatus(status as Integer) as Boolean
```

status

Identifies the value that is to be set.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

This script shows the use of the SetStatus function.

```
if MsgBox("Should this job return an error?",MB_YESNO)=IDYES then
        SetStatus(NC_ERROR)
else
        SetStatus(NC_OK)
        Print ("The user selected OK when prompted...")
end if
```

Sleep - Miscellaneous Function

Sleep is an alias for Delay function.

SortArray - Sort an Array

Valid on UNIX and Windows

The SortArray function sorts an array.

Function format:

SortArray(array as array) as Boolean

SortArray(array as array, start as integer) as Boolean

SortArray(array as array, start as integer, end as integer) as Boolean

Array

Identifies the name of the array to be sorted.

Start

Specifies the start index for the sort. The array is sorted from the start index to the end of the array.

End

Specifies the last index for sort.

The function only handles basic types of arrays. Depending on the format chosen, (1) the whole array is sorted, (2) the array is sorted beginning with the start index, and (3) the array is sorted from the start index to the end index.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
Dim a[10], b[10], i As Integer
Dim TAB As Char
TAB = chr(9)

For i=0 to 9
        a[i] = 10 - i
Next i

b = a
If SortArray(b) Then
        For i = 0 To 9
                Print("a("+Str(i)+")="+Str(a[i])+TAB+"b("+Str(i)+")="+Str(b[i]))
        Next i
Else
        Print("SortArray(b) failed!")
End If
```

Handling Backward Compatibility or Unknown Functions

DMScript provides functions that handle new functions on an old interpreter or unknown functions within a script. The following functions help ensure that the newer scripts run successfully on older DSM agents by ignoring the new or unknown functions:

- ALLOW_UNRESOLVED_FUNCTIONS
- IsFunctionSupported

ALLOW_UNRESOLVED_FUNCTIONS

The IsFunctionSupported function checks whether the current version of dmscript understands a given function. This directive has the following format:

```
#pragma ALLOW_UNRESOLVED_FUNCTIONS
```

When you add this directive to a DMScript, the compiler performs the following action on finding an unknown function in the script:

- Adds a warning to the trace log at compile time
- Emits code that outputs a fatal runtime error when the script attempts to execute the unknown function at run time.

IsFunctionSupported

The IsFunctionSupported function checks whether a function is supported.

Function format:

```
IsFunctionSupported (name as string) as Boolean
```

Example:

```
#pragma ALLOW_UNRESOLVED_FUNCTIONS
If IsFunctionSupported ("OpenDetectedSoftwareOutputFiles") then
OpenDetectedSoftwareOutputFiles ("A7C1E14A-7C93-4E17-B4E5-45B796717F49", "V1", "OS Detection for
Windows")
else
   dsmtrace("error","This version of dmscript does not support Intellisigs.")
   exit
end if
```

Input Parameters

This function has the following input parameter:

name

Specifies the name of the function.

Return Values

If the given function is either defined in the script or a built-in function, the function returns true; else, returns false. This function is used with the ALLOW_UNRESOLVED_FUNCTIONS pragma to allow scripts to handle backward compatibility. If a script uses a function that exists in the current release but not in previous releases, the script fails compilation and does not perform any action on computers running the old releases. IsFunctionSupported allows such a script to pass compilation on old releases and perform a special action at runtime to handle the missing feature. It is not possible to compensate for the missing function but an error message is displayed for the user.

Note: You can also use the GetFileInfo (see page 113) and CompareVersions (see page 115) functions to verify that the script is executing on a particular version of DMScript.

Network Functions

NetGetMap - Retrieve a Network Resource Name

Valid on Windows and Windows CE

The NetGetMap function retrieves the name of the network resource associated with a redirected device specified in device.

Function format:

```
NetGetMap(device as String) as String
```

device

Identifies the drive or printer specification. Valid values are A: through Z: and LPT1: through LPT3:

The function returns a string containing the name of the network resource to which a device was redirected. If the device is not redirected, the function returns an empty string.

Example:

This example maps the AM sector share, displays the share and un-maps it again.

```
If SectorService <> "" Then
 If NetMap("U:",SectorService) Then
  MessageBox("U: Is Mapped to " + NetGetMap("U:"))
  NetUnMap("U:")
 Else
  MessageBox("Could not map Sector Share")
 Endif
Else
 MessageBox("Could not find Sector Share Name")
EndIf
```

NetMap - Map a Local Device to a Network Resource

Valid on Windows and Windows CE

The NetMap function maps a local device to a network resource.

Function format:

```
NetMap(device as String, service as String, password as String) as Boolean
```

```
NetMap(device as String, service as String) as Boolean
```

device

Specifies the local drive or device to be redirected. (Valid values are A: through Z: and LPT1: through LPT3:)

service

Specifies the name of the network resource to which the local device is to be redirected.

password

Identifies an optional string specifying the password for the network resource.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

This example maps the AM sector share, displays the share and un-maps it again.

```
If SectorService <> "" Then
 If NetMap("U:",SectorService) Then
  MessageBox("U: Is Mapped to " + NetGetMap("U:"))
  NetUnMap("U:")
 Else
  MessageBox("Could not map Sector Share")
 Endif
Else
 MessageBox("Could not find Sector Share Name")
EndIf
```

NetUnMap - Cancel the Redirection

Valid on Windows

The NetUnMap function cancels the redirection of a redirected device.

Function format:

```
NetUnMap(device as String) as Boolean

NetUnMap(device as String, force as Boolean) as Boolean
```

device

Identifies the redirected drive or device. (Valid values are A: through Z: and LPT1: through LPT3:)

force

Specifies whether any open files on the device are to be closed before the connection is canceled. If the force parameter is FALSE and there are open files on the redirected device, the function fails.

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

This example maps the AM sector share, displays the share and un-maps it again.

```
If SectorService <> "" Then
 If NetMap("U:",SectorService) Then
  MessageBox("U: Is Mapped to " + NetGetMap("U:"))
  NetUnMap("U:")
 Else
  MessageBox("Could not map Sector Share")
 Endif
Else
 MessageBox("Could not find Sector Share Name")
EndIf
```

Registry Manipulation Functions

RegCloseKey - Close the Registry Key

Valid on Windows

The RegCloseKey function closes the registry key associated with the hKey parameter, a handle obtained from a call of RegOpenKey or RegCreateKey.

Function format:

```
RegCloseKey(hKey as Integer) as Boolean
```

hkey

hKey value of registry key.

On successful completion, the function returns TRUE. If the function fails (hKey is not the handle of a valid, open registry key), the function returns FALSE.

Example:

```
Dim hkey1, hkey2, hkey3 as integer

Dim cBuf as string
Dim ccBuf[47] as char
Dim bBuf[100] as byte
Dim i as integer

ClrScr()
for i = 0 to 99
        bBuf[i] = i
next i
cBuf = "1234567890ßqwertzuiopü+asdfghjklöä#<yxcvbnm,.-"
ccBuf = cBuf

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

hkey2 = RegCreateKey(hkey1, "CA")
if hkey2 = 0 then
        Print("RegCreateKey 1 failed.")
        exit
endif

if Not(RegSetValue(hkey1, "CA", "GmbH Paderborn - Germany")) then

Print("RegSetValue failed")
endif

hkey3 = RegCreateKey(hkey2, "4 test only")
if hkey3 = 0 then
        Print("RegCreateKey 2 failed.")
        exit
endif
```

```
if Not(RegSetVariable(hkey3, "var_1", 123)) then
        Print("RegSetVar 1 failed.")
endif
if Not(RegSetVariable(hkey3, "var_2", "i'm a string")) then
        Print("RegSetVar 2 failed.")
endif
if Not(RegSetVariable(hkey3, "var_3", cBuf, 46)) then
        Print("RegSetVar 3 failed.")
endif
if Not(RegSetVariable(hkey3, "var_4", bBuf, 100)) then

Print("RegSetVar 4 failed.")
endif
if Not(RegSetVariable(hkey3, "var_5", cBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif
if Not(RegSetVariable(hkey3, "var_6", ccBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif

if Not(RegCloseKey(hkey3)) then
        Print("RegCloseKey for hkey3 failed.")
endif
if Not(RegCloseKey(hkey2)) then
        Print("RegCloseKey for hkey2 failed.")
endif
if Not(RegCloseKey(hkey1)) then
        Print("RegCloseKey for hkey1 failed.")


endif
```

RegCreateKey - Create a Registry key

Valid on Windows

The RegCreateKey function creates a specific registry key. If the key already exists in the registry, it is opened.

Function format:

```
RegCreateKey(hKey as Integer, subkey as String) as Integer
```

hKey

Identifies a currently open key or a predefined key. The hKey parameter can be any of the following predefined values available on all Windows platforms:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

subkey

Specifies a string that indicates the name of the key to open. Subkey must be a subkey of the key identified by hKey. If the subkey parameter is an empty string, the function returns the same key as that specified in the hKey parameter.

On successful completion, the function returns a handle to the newly created registry key; otherwise, returns 0.

Example:

```
Dim hkey1, hkey2, hkey3 as integer
Dim cBuf as string
Dim ccBuf[47] as char
Dim bBuf[100] as byte
Dim i as integer

ClrScr()
for i = 0 to 99
        bBuf[i] = i
next i
cBuf = "1234567890ßqwertzuiopü+asdfghjklöä#<yxcvbnm,.-"
ccBuf = cBuf

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

hkey2 = RegCreateKey(hkey1, "CA Technology")
if hkey2 = 0 then
        Print("RegCreateKey 1 failed.")
        exit
endif

if Not(RegSetValue(hkey1, "CA", "International")) then
        Print("RegSetValue failed")
endif

hkey3 = RegCreateKey(hkey2, "4 test only")
if hkey3 = 0 then
        Print("RegCreateKey 2 failed.")
        exit
endif

if Not(RegSetVariable(hkey3, "var_1", 123)) then
        Print("RegSetVar 1 failed.")
endif
if Not(RegSetVariable(hkey3, "var_2", "i'm a string")) then
        Print("RegSetVar 2 failed.")
endif
if Not(RegSetVariable(hkey3, "var_3", cBuf, 46)) then
        Print("RegSetVar 3 failed.")
endif
```

```
if Not(RegSetVariable(hkey3, "var_4", bBuf, 100)) then
        Print("RegSetVar 4 failed.")
endif
if Not(RegSetVariable(hkey3, "var_5", cBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif
if Not(RegSetVariable(hkey3, "var_6", ccBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif

if Not(RegCloseKey(hkey3)) then
        Print("RegCloseKey for hkey3 failed.")
endif
if Not(RegCloseKey(hkey2)) then
        Print("RegCloseKey for hkey2 failed.")
endif
if Not(RegCloseKey(hkey1)) then
        Print("RegCloseKey for hkey1 failed.")

endif
```

RegDeleteKey - Delete a Registry Key

Valid on Windows

The RegDeleteKey function removes the specified key.

Function format:

```
RegDeleteKey(hKey as Integer, subkey as String) as Boolean
```

hKey

Indicates a currently open key or a predefined key and can be any of the following predefined values, which are available on all Windows platforms:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

subkey

Indicates the parameter is a string that specifies the name of the key to delete. Subkey must be a subkey of the key identified by hKey. Do not use an empty string as a subkey parameter.

On successful completion, the function returns TRUE. If the key does not exist or is currently opened by another application, it returns FALSE.

subkeys

On Windows 9x, this function also removes all subkeys. On Windows NT and 2000, you must remove each subkey separately before running the function.

Example:

Test structure:

HKLM\

\ Software

\ CA

\ 4 test only

\ var_6

\ dummy

```
Dim hkey1 as integer

ClrScr()


hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA\4 test only")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteValue(hKey1) then
        Print("RegDeleteValue succeeded.")
else
        Print("RegDeleteValue failed.")
endif

if RegDeleteVariable(hKey1, "var_6") then
        Print("RegDeleteVariable for var_6 succeeded.")
else
        Print("RegDeleteVariable for var_6 failed.")
endif
```

```
RegCloseKey(hkey1)

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteKey(hkey1, "dummy") then
        Print("Key dummy successfully deleted")
else
        Print("Key dummy deletion failed.")
endif

RegCloseKey(hkey1)

Messagebox("Check registry!", "desktop management scripting test")

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteKey(hkey1, "CA\4 test only") then
        Print("Key ""CA\4 test only"" successfully deleted")

else
        Print("Key" "CA\4 test only" "deletion failed.")
endif

if RegDeleteKey(hkey1, "CA") then
        Print("Key CA successfully deleted")
else
        Print("Key CA deletion failed.")
endif


RegCloseKey(hkey1)
```

RegDeleteValue - Delete a Value in the Registry

Valid on Windows

The RegDeleteValue function deletes a value in the registry.

Function format:

```
RegDeleteValue(hkey as integer) as Boolean
```

hKey

Specifies the handle of the registry key from which to remove the associated value.

The return value is TRUE if the function succeeds; otherwise, FALSE.

Example:

Test structure:

HKLM\

\ Software

\ CA

\ 4 test only

\ var_6

\ dummy

```
Dim hkey1 as integer

ClrScr()


hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA\4 test only")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif
```

```
if RegDeleteValue(hKey1) then
        Print("RegDeleteValue succeeded.")
else
        Print("RegDeleteValue failed.")
endif

if RegDeleteVariable(hKey1, "var_6") then
        Print("RegDeleteVariable for var_6 succeeded.")
else
        Print("RegDeleteVariable for var_6 failed.")
endif

RegCloseKey(hkey1)

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteKey(hkey1, "dummy") then
        Print("Key dummy successfully deleted")
else
        Print("Key dummy deletion failed.")
endif

RegCloseKey(hkey1)

Messagebox("Check registry!", "desktop management scripting test")

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteKey(hkey1, "CA\4 test only") then
        Print("Key ""CA\4 test only"" successfully deleted")

else
        Print("Key" "CA\4 test only" "deletion failed.")
endif
```

```
if RegDeleteKey(hkey1, "CA") then
        Print("Key CA successfully deleted")
else
        Print("Key CA deletion failed.")
endif


RegCloseKey(hkey1)
```

RegDeleteVariable - Remove a Variable

Valid on Windows

The RegDeleteVariable function removes the specified variable.

Function format:

```
RegDeleteVariable(hKey as integer, name as string) as Boolean
```

hKey

Indicates a currently opened key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

name

Identifies the name of the variable that is to be removed.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

Test structure:

HKLM\

\ Software

\ CA

\ 4 test only

\ var_6

\ dummy

```
Dim hkey1 as integer

ClrScr()


hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA\4 test only")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteValue(hKey1) then
        Print("RegDeleteValue succeeded.")
else
        Print("RegDeleteValue failed.")
endif

if RegDeleteVariable(hKey1, "var_6") then
        Print("RegDeleteVariable for var_6 succeeded.")
else
        Print("RegDeleteVariable for var_6 failed.")
endif

RegCloseKey(hkey1)

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif
```

```
if RegDeleteKey(hkey1, "dummy") then
        Print("Key dummy successfully deleted")
else
        Print("Key dummy deletion failed.")
endif

RegCloseKey(hkey1)

Messagebox("Check registry!", "desktop management scripting test")

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

if RegDeleteKey(hkey1, "CA\4 test only") then
        Print("Key ""CA\4 test only"" successfully deleted")

else
        Print("Key" "CA\4 test only" "deletion failed.")
endif

if RegDeleteKey(hkey1, "CA") then
        Print("Key CA successfully deleted")
else
        Print("Key CA deletion failed.")
endif


RegCloseKey(hkey1)
```

RegEnumKey - Enumerate the Subkeys of a Registry Key

Valid on Windows

The RegEnumKey function enumerates the subkeys of a specified registry key.

Function format:

RegEnumKey(hKey as Integer, index as Integer, subkey as String) as Boolean

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

index

Identifies the index of the subkey that is to be retrieved. Use zero for the first call of the function.

subkey

Identifies the output parameter that is to hold the requested subkey.

On successful completion, the function returns TRUE and fills in the name of the subkey in the subkey variable. If the key is invalid or there are no more subkeys, it returns FALSE.

Example:

Presupposed registry structure:

HKLM

Software

CA

test-1

...

test-9

dummy

4test only

var-0 "test 2"

...

```
Rem
Rem This example enumerates the subkeys of HKEY_CLASSES_ROOT,
Rem printing all filetype entries.
Rem

Dim SubKey as string
Dim Index as integer

Index=0
while RegEnumKey(HKEY_CLASSES_ROOT,Index,SubKey)
        if Left(SubKey,1)="." then print SubKey
        Index=Index+1
wend
```

RegEnumVariable - Enumerate the Variables

Valid on Windows

The RegENumVariable function enumerates the variables for the specified registry key.

Function format:

RegEnumVariable(hKey as Integer, index as Integer, name as String, strvalue as String, intvalue as Integer) as Integer

RegEnumVariable(hkey as integer, index as integer, dummy as string, name as string, bufsize as integer, buffer as void) as integer.

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values available on all Windows platforms:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

index

Identifies the index of the variable that is to be retrieved. Use zero for the first call of the function.

name

Identifies the output parameter that is to hold the name of the requested variable.

strvalue

Identifies the output parameter that is to hold the string value of the requested variable.

intvalue

Identifies the output parameter that is to hold the integer value of the requested variable.

dummy

Reserved.

bufsize

Size of the receiving buffer in chars if the buffer is of type string; otherwise, the number of bytes are specified. Must be a variable. After a successful return, it contains the number of bytes retrieved.

buffer

Buffer to receive the retrieved value of the variable.

On successful completion, the first variant of the function returns one of the following values:

REG_NODATA

Value 0; A variable was not found. This indicates that either the last index of the enumeration has been reached or the handle was invalid.

REG_INTEGER

Value 1; An Integer type variable was found; the value will be stored in the intvalue variable.

REG_STRING

Value 2; A String type variable was found; the value will be stored in the strvalue variable.

REG_UNSUPPORTED

Value 3; An unsupported variable type was found; the value will be void.

The second variant of the function returns in Microsoft notation:

REG_NONE

Value 0

REG_SZ

Value 1

REG_EXPAND_SZ

Value 2

REG_BINARY

Value 3

REG_DWORD

Value 4

REG_DWORD_LITTLE_ENDIAN

Value 4

REG_DWORD_BIG_ENDIAN

Value 5

REG_LINK

Value 6

REG_MULTI_SZ

Value 7

Example:

```
Dim value as string
Dim hkey1 as integer
Dim name, str, dummy as string
Dim i, i1, rtr, int as integer
Dim bBuf[100] as Byte
Dim cBuf[100] as char

ClrScr()

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif
```

```
i = 0
while
        RegEnumKey(hkey1, i, str)
        Print(Str(i) + " - " + str)

RegCloseKey(hkey1)

Print " "

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA\4 test only")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif
for i = 0 to 1
        rtr = RegEnumVariable(hKey1, i, name, str, i1)
        if (REG_INTEGER = rtr) then
                Print(Str(rtr) + ": " + Str(i) + " - " + name + " = " + Str(i1))
        else
                if (REG_STRING = rtr) then
                        Print(Str(rtr) + ": " + Str(i) + " - " + name + " = " + str)
                else
                        Print(Str(rtr) + ": " + "unknown type");
                endif
        endif
next i
i1 = 100
str = ""
rtr = RegEnumVariable(hKey1, 2, name, dummy, i1, bBuf))
for i = 0 to i1 - 1
        str = str + Str(bBuf[i])
next i
Print (Str(rtr) + ": " + "2 - " + name + "(" + Str(i1) + ") = " + str)
```

RegOpenKey - Open a Key

Valid on Windows

The RegOpenKey function opens a specified registry key.

Function format:

```
RegOpenKey(hKey as Integer, subkey as String) as Integer
```

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

subkey

Identifies the key to be opened. Subkey key must be a subkey of the key identified by the hKey parameter. If subkey is an empty string, the function returns the same key as that specified by hKey.

On successful completion, the function returns a handle to the newly opened registry key; otherwise, returns zero.

Example:

```
Dim hkey1, hkey2, hkey3 as integer

Dim cBuf as string
Dim ccBuf[47] as char
Dim bBuf[100] as byte
Dim i as integer

ClrScr()
for i = 0 to 99
        bBuf[i] = i
next i
cBuf = "1234567890ßqwertzuiopü+asdfghjklöä#<yxcvbnm,.-"
ccBuf = cBuf

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

hkey2 = RegCreateKey(hkey1, "CA")
if hkey2 = 0 then
        Print("RegCreateKey 1 failed.")
        exit
endif
```

```
if Not(RegSetValue(hkey1, "CA", "GmbH Paderborn - Germany")) then

Print("RegSetValue failed")
endif

hkey3 = RegCreateKey(hkey2, "4 test only")
if hkey3 = 0 then
        Print("RegCreateKey 2 failed.")
        exit
endif

if Not(RegSetVariable(hkey3, "var_1", 123)) then
        Print("RegSetVar 1 failed.")
endif
if Not(RegSetVariable(hkey3, "var_2", "i'm a string")) then
        Print("RegSetVar 2 failed.")
endif
if Not(RegSetVariable(hkey3, "var_3", cBuf, 46)) then
        Print("RegSetVar 3 failed.")
endif
if Not(RegSetVariable(hkey3, "var_4", bBuf, 100)) then

Print("RegSetVar 4 failed.")
endif
if Not(RegSetVariable(hkey3, "var_5", cBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif
if Not(RegSetVariable(hkey3, "var_6", ccBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif

if Not(RegCloseKey(hkey3)) then
        Print("RegCloseKey for hkey3 failed.")
endif
if Not(RegCloseKey(hkey2)) then
        Print("RegCloseKey for hkey2 failed.")
endif
if Not(RegCloseKey(hkey1)) then
        Print("RegCloseKey for hkey1 failed.")

endif
```

RegQueryValue - Retrieve a Text String

Valid on Windows

The RegQueryValue function retrieves the text string associated with the specified registry key\subkey, if the unnamed default value (data) of the opened key is to be fetched.

Function format:

RegQueryValue(hKey as Integer, subkey as String, value as String) as Boolean

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

subkey

Identifies, together with the key, the default value that is to be retrieved. The subkey can be any subkey or the empty string, if the unnamed value of the opened key is to be fetched. The subkey must be a subkey of the key identified by the hKey parameter. The parameter value is a string variable that receives the value of the subkey.

Value

Holds the returned value (data) of the specified key\subkey.

On successful completion, the function returns TRUE and fills the string associated with the specified key\subkey into the value variable; otherwise, returns FALSE.

RegQueryVariable - Retrieve the Type and Value of a Variable

Valid on Windows

The RegQueryVariable function retrieves the type and value of the specified variable for the specified registry key.

Function format:

Format 1

RegQueryVariable(hKey as Integer, name as String, strvalue as String, intvalue as Integer) as Integer

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

name

Identifies the name of the requested variable.

strvalue

Identifies the output parameter that is to hold the string value of the requested variable.

intvalue

Identifies the output parameter that is to hold the integer value of the requested variable.

On successful completion, the function returns one of the following values:

REG_NODATA

A variable was not found. This indicates that the handle was invalid.

REG_INTEGER

An integer type variable was found; the value will be stored in the intvalue variable.

REG_STRING

A string type variable was found; the value will be stored in the strvalue variable.

REG_UNSUPPORTED

An unsupported variable type was found; the value will be void.

Format 2

RegQueryVariable(hkey as integer, name as string, dummy as string, bufsize as integer, buffer as void) as integer.

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

name

Identifies the name of the requested variable.

dummy

Reserved.

bufsize

Size of the receiving buffer in chars if the buffer is of type string; otherwise, the number of bytes are specified. Must be a variable. After successful return it contains the number of bytes retrieved.

buffer

Buffer to receive the retrieved value of the variable.

Type of registry entry is returned in Microsoft notation:

REG_NONE

Value 0

REG_SZ

Value 1

REG_EXPAND_SZ

Value 2

REG_BINARY

Value 3

REG_DWORD

Value 4

REG_DWORD_LITTLE_ENDIAN

Value 5

REG_DWORD_BIG_ENDIAN

Value 6

REG_LINK

Value 7

REG_MULTI_SZ

Value 8

Example:

Before you start the following example, run the RegOpenKey example.

```
Dim value as string
Dim hkey1 as integer
Dim name, str, dummy as string
Dim i, i1, rtr, int as integer
Dim bBuf[100] as Byte
Dim cBuf[100] as char

ClrScr()

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA\4 test only")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

rtr = RegQueryVariable(hkey1, "var_2", str, int)
Print(Str(rtr) + ": var_2 = " + str)
int = 100
str = ""
rtr = RegQueryVariable(hkey1, "var_2", dummy, int, str)
Print(Str(rtr) + ": var_2(" + Str(int) + ") = " + str)
int = 100
str = ""
rtr = RegQueryVariable(hkey1, "var_2", dummy, int, cBuf)
for i1 = 0 to int - 1
        str = str + cBuf[i1]
next i1
Print(Str(rtr) + ": var_2(" + Str(int) + ") = " + str)
int = 100
str = "| "
rtr = RegQueryVariable(hkey1, "var_2", dummy, int, bBuf)
for i1 = 0 to int - 1
        str = str + Str(bBuf[i1]) + " | "
next i1
Print(Str(rtr) + ": var_2(" + Str(int) + ") = " + str)

RegCloseKey(hkey1)
```

RegSetValue - Associate the Registry Key with a Text String

Valid on Windows

The RegSetValue function associates the specified registry key with a text string.

Function format:

RegSetValue(hKey as Integer, subkey as String, value as String) as Boolean

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

subkey

Identifies the name of the subkey that is to be set.

value

Identifies the String value that is to be set in the variable.

On successful completion, the function returns TRUE; otherwise, returns FALSE.

Example:

```
Dim hkey1, hkey2, hkey3 as integer
Dim cBuf as string
Dim ccBuf[47] as char
Dim bBuf[100] as byte
Dim i as integer

ClrScr()
for i = 0 to 99
        bBuf[i] = i
next i
cBuf = "1234567890ßqwertzuiopü+asdfghjklöä#<yxcvbnm,.-"
ccBuf = cBuf

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE")

if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

hkey2 = RegCreateKey(hkey1, "CA Technology")
if hkey2 = 0 then
        Print("RegCreateKey 1 failed.")
        exit
endif

if Not(RegSetValue(hkey1, "CA", "International")) then
        Print("RegSetValue failed")
endif

hkey3 = RegCreateKey(hkey2, "4 test only")
if hkey3 = 0 then
        Print("RegCreateKey 2 failed.")
        exit
endif

if Not(RegSetVariable(hkey3, "var_1", 123)) then
        Print("RegSetVar 1 failed.")
endif
if Not(RegSetVariable(hkey3, "var_2", "i'm a string")) then
        Print("RegSetVar 2 failed.")
endif
if Not(RegSetVariable(hkey3, "var_3", cBuf, 46)) then
        Print("RegSetVar 3 failed.")
endif
if Not(RegSetVariable(hkey3, "var_4", bBuf, 100)) then
        Print("RegSetVar 4 failed.")
endif
```

```
if Not(RegSetVariable(hkey3, "var_5", cBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif
if Not(RegSetVariable(hkey3, "var_6", ccBuf, 46, REG_SZ)) then
        Print("RegSetVar 5 failed.")
endif

if Not(RegCloseKey(hkey3)) then
        Print("RegCloseKey for hkey3 failed.")
endif
if Not(RegCloseKey(hkey2)) then
        Print("RegCloseKey for hkey2 failed.")
endif
if Not(RegCloseKey(hkey1)) then
        Print("RegCloseKey for hkey1 failed.")

endif
```

RegSetVariable - Store a Variable in the Registry Key

Valid on Windows

The RegSetVariable function stores a variable in the specified registry key.

Function format:

RegSetVariable(hKey as Integer, name as String, value as Integer) as Boolean

RegSetVariable(hKey as Integer, name as String, value as String) as Boolean

RegSetVariable(hkey as integer, name as string, buffer as void, bufsize as integer) as integer.

RegSetVariable(hkey as integer, name as string, buffer as void, bufsize as integer, type as integer) as integer.

hKey

Identifies a currently open key or a predefined key. The hKey parameter can have one of the following predefined values:

HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

HKEY_LOCAL_MACHINE

HKEY_USERS

name

Identifies the name of the variable that is to be set.

value

Identifies the string or Integer value that is to be set in the variable.

buffer

Buffer containing the value to be set in the variable.

bufsize

Size of the buffer in chars if the buffer is of type string or array of type char; otherwise, the number of bytes are specified.

type

The type of registry entry.

Type of registry entry is returned in Microsoft notation:

REG_NONE

Value 0

REG_SZ

Value 1

REG_EXPAND_SZ

Value 2

REG_BINARY

Value 3

REG_DWORD

Value 4

REG_DWORD_LITTLE_ENDIAN

Value 5

REG_DWORD_BIG_ENDIAN

Value 6

REG_LINK

Value 7

REG_MULTI_SZ

Value 8

On successful completion, the function returns TRUE; otherwise, it returns FALSE.

Example:

Before you start the following example, run the RegOpenKey example.

```
Dim value as string
Dim hkey1 as integer
Dim name, str, dummy as string
Dim i, i1, rtr, int as integer
Dim bBuf[100] as Byte
Dim cBuf[100] as char

ClrScr()

hkey1 = RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\CA\4 test only")
if hkey1 = 0 then
        Print("RegOpenKey failed.")
        exit
endif

rtr = RegQueryVariable(hkey1, "var_2", str, int)
Print(Str(rtr) + ": var_2 = " + str)
int = 100
str = ""
rtr = RegQueryVariable(hkey1, "var_2", dummy, int, str)
Print(Str(rtr) + ": var_2(" + Str(int) + ") = " + str)
int = 100
str = ""
rtr = RegQueryVariable(hkey1, "var_2", dummy, int, cBuf)
for i1 = 0 to int - 1
        str = str + cBuf[i1]
next i1
Print(Str(rtr) + ": var_2(" + Str(int) + ") = " + str)
int = 100
str = "| "
rtr = RegQueryVariable(hkey1, "var_2", dummy, int, bBuf)
for i1 = 0 to int - 1
        str = str + Str(bBuf[i1]) + " | "
next i1
Print(Str(rtr) + ": var_2(" + Str(int) + ") = " + str)

RegCloseKey(hkey1)
```

SetMode64 - Accessing Windows 64-bit Registry and File System

DMScript functions can access the Windows registry and launch programs. When DMScript runs on a 64-bit Window, the OS imposes certain rules as DMScript is a 32-bit application. Registry access for certain keys is automatically redirected to a parallel 32-bit registry area. For example, access to HKLM\SOFTWARE\xxx is redirected to HKLM\SOFTWARE\Wow6432Node\xxx. For a complete list of redirected keys, see: http://msdn.microsoft.com/en-us/library/aa384253(v=vs.85).aspx.

Similarly, accessing the file system using certain environment variables that reference system directories, are automatically redirected to directories that contain 32-bit binaries. For example, the %windir%\System32 directory is reserved for 64-bit applications. If DMScript attempts to access this directory, it is automatically redirected to %windir%\SysWOW64. For more information, see http://msdn.microsoft.com/en-us/library/aa384187(v=vs.85).aspx.

This behavior is important for Intellisig scripts because the scripts must detect 64-bit applications by accessing the correct 64-bit registry keys and file systems. For example, if an Intellisig attempts to read the registry key for the 64-bit version of 7zip in HKLM\SOFTWARE\7-zip, the script fails because it actually searches in HKLM\SOFTWARE\ Wow6432Node\7-zip does not exist. Similarly, an attempt to look for the 64-bit version of Notepad in %windir%\System32 finds the 32-bit version.

To handle this behavior, DMScript provides the following function that turns off the automatic redirection on 64-bit Windows OS:

setmode64(mode)

Mode

Specifies whether the redirection must be turned off or on. A value of 1 turns off redirection and a value of 0 turns it on.
Example: setmode64(1)

Note: When in the 64-bit mode, access the 32-bit registry using the physical name such as HKLM\SOFTWARE\Wow6432Node\xxx. However, Microsoft does not recommend this practice because the method of redirection can change in the future versions of Windows. Instead, the script must revert to the 32-bit mode.

Important! Disabling file system redirection for longer periods can prevent dmscript from loading system DLLs, causing it to fail.

String Manipulation Functions

The String Manipulation functions can be used on all supported operating systems.

Important! The input strings for String Manipulation Functions must not contain Japanese characters on Linux computers.

Asc - Convert a Character to an ASCII Value

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The asc function converts a character to its corresponding ASCII value or the wide character code of a glyph. If parameter s1 is a string and contains more than one character, the function evaluates the first character only.

Function format:

```
Asc(s1 as String) as Integer

Asc(s1 as Char) as Integer
```

s1

Identifies name of character

The return value is the wide character code value of the input character or the first character of the input string.

Example:

```
Dim OutStr as string

OutStr = Str(Asc("A"))+" "+Str(Asc("B"))+" "+Str(Asc("C"))

Print "AscII Values of the letters in ""ABC"" is: " + OutStr
```

Chr - Convert Integer to Wide Character

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Chr function converts an integer value to its corresponding wide character.

Function format:

```
Chr(n1 as Integer) as Char
```

n1

Identifies integer value

The function returns one wide character specified by the n1 parameter.

Example:

```
Print "LITTLE "+Chr(65)+Chr(66)+Chr(67)
```

InStr - Scan a String for the occurrence of a Substring

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The InStr function scans a string for the occurrence of a given substring. The function scans the string str for the first occurrence of the substring substr.

Function format:

```
Instr(str as String, substr as String) as Integer
```

str

Identifies the string for which a search is being performed.

substr

Identifies the substring for which a search is being performed.

The function returns the character position in str, at which substr begins. If substr does not occur in str, the function returns 0.

Example:

This example uses the InStr function to determine if the word "Script" is a part of the sentence "Unicenter Asset Management Script Interpreter".

```
Dim MyStr as String ' Contains string to evaluate
Dim Pos as Integer ' Holds a string index

MyStr="Unicenter Asset Management Script Interpreter"

Pos=InStr(MyStr,"Script")

if Pos>0 then
        Print "The word '""Script""' was found at character position "+str(Pos)+" in"
        Print "the sentence '""+MyStr+'""'."
else

        Print "The word '""Script""' was NOT found in"
        Print "the sentence '""+MyStr+'""'."
End if
```

LCase - Convert Uppercase Letters to Lowercase Letters

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The LCase function converts uppercase letters in the s1 string to lowercase letters. No other characters are affected. There is no restriction to the ASCII code table.

Function format:

```
Lcase (s1 as String) as String
```

s1

Identifies the string to be converted.

The function returns the converted string.

Example:

This example displays a text in uppercase and lowercase.

```
Rem
Dim text As String
Text="Little ABC"

Print("LCase : " + LCase(text)) 'little abc

Print("UCase : " + UCase(text)) 'LITTLE ABC
```

Left - Return the Leftmost n Characters from the String

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Left function returns a substring containing the leftmost n characters from the string str.

Function format:

```
Left(str as String, n as Integer) as String
```

str

Identifies the string from which the left substring is to be taken.

n

Indicates the number of characters of the left substring.

The function returns a substring containing the leftmost *n* characters from the string str.

Example: Left function

This example shows how the string operations work.

```
Dim text As String
Text="Unicenter TNG"

Print("Left(text,3) : " + Left(text,3))  'Uni
Print("Mid(text,4,6) : " + Mid(text,4,6)) 'center
Print("Right(text,3) : " + Right(text,3)) 'TNG
Print("Length    : " + Str(Len(text))) '13
```

Len - Obtain the Length of the String

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Len function obtains the length of the string str.

Function format:

```
Len(str as String) as Integer
```

str

Identifies the specified string.

This function returns the number of characters in the string str.

Example:

This example shows how the string operations work.

```
Dim text As String
Text="Unicenter TNG"

Print("Left(text,3) : " + Left(text,3))  'Uni
Print("Mid(text,4,6) : " + Mid(text,4,6)) 'center
Print("Right(text,3) : " + Right(text,3)) 'TNG
Print("Length    : " + Str(Len(text))) '13
```

Mid - Return a Substring from a String

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Mid function returns a substring from a string.

Function format:

```
Mid(str as String, start as Integer, length as Integer) as String
```

```
Mid (str as String, start as Integer) as String
```

str

Identifies the string from which a substring is taken.

start

Indicates the start position of the substring in str.

len

Indicates the length of the substring.

Note: The arguments start and the length must be between 1 and 65,535, inclusive. If length is omitted or if there are fewer than *n* length characters in str, the function returns all characters from the start position to the end of the string. If start is greater than the number of characters in str, the function returns a zero length string.

The return value of the function is a substring from the string str as described previously.

Example:

This example shows how the string operations work.

```
Dim text As String
Text="Unicenter TNG"

Print("Left(text,3) : " + Left(text,3))  'Uni
Print("Mid(text,4,6) : " + Mid(text,4,6)) 'center
Print("Right(text,3) : " + Right(text,3)) 'TNG
Print("Length    : " + Str(Len(text))) '13
```

Right - Return a Substring with Rightmost n Characters

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Right function returns a substring containing the rightmost n characters from the string str.

Function format:

```
Right(str as String, n as Integer) as String
```

str

Indicates the string from which the right substring is to be taken.

n

Specifies the number of characters of the substring.

The function returns a substring containing the rightmost n characters from the string str.

Example:

This example shows how the string operations work.

```
Dim text As String
Text="Unicenter TNG"

Print("Left(text,3) : " + Left(text,3))  'Uni
Print("Mid(text,4,6) : " + Mid(text,4,6)) 'center
Print("Right(text,3) : " + Right(text,3)) 'TNG
Print("Length    : " + Str(Len(text))) '13
```

Str - Convert Numeric Expression to String

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Str function converts the value of a numeric expression to a string.

Function format:

```
Str(n as Integer) as String
```

n

Identifies the integer that is to be converted.

The function returns the string representing the value specified by the parameter n.

Example:

This example prints the ASCII values of the letters "ABC" on the screen.

```
Dim OutStr as string

OutStr = Str(Asc("A"))+" "+Str(Asc("B"))+" "+Str(Asc("C"))

Print "AscII Values of the letters in ""ABC"" is: " + OutStr
```

Trim - Remove Leading and Trailing Spaces from a String

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Trim function removes leading and trailing spaces from the string.

Function format:

```
Trim(str as String) as String
```

str

Identifies the string that is to be trimmed.

The function returns the trimmed string.

Example:

```
Dim tStr, str As String

str = "  abcd "
tStr = Trim(str)
Print(">>>"+Str+"<<<")
Print(">>>"+tStr+"<<<")

This prints:
>>>  abcd <<<
>>>abcd<<<
```

UCase - Convert Lowercase Letter to Uppercase

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The UCase function converts lowercase letters in the s1 string to uppercase letters. No other characters are affected. There is no restriction to the ASCII code table.

This string manipulation function has the format:

```
UCase(s1 as String) as String
```

s1

Identifies the string that is to be converted.

The function returns the converted string.

Example:

This example displays a text in uppercase and lowercase.

```
Rem
Dim text As String
Text="Little ABC"

Print("LCase : " + LCase(text)) 'little abc

Print("UCase : " + UCase(text)) 'LITTLE ABC
```

Val - Convert String to Integer

Valid on NetWare, Symbian OS, UNIX, Windows and Windows CE

The Val function converts the input string to an integer.

Function format:

```
Val(str as String) as Integer
```

str

Identifies the string that is to be converted.

This function ignores any leading tabs or spaces and recognizes an optional sign followed by a string of digits.

The function returns the converted value of the input string. If the string cannot be converted to a number, the function returns 0.

Example:

```
Dim i As Integer
i = Val("2735")
Print("Val(""2735"")"+Str(i))
i=ValEx("2735")
Print("ValEx(2735")
Print("ValEx(""2735"")="+str(i))
i=ValEx("01414")
Print("ValEx(""01414"")="+str(i))
i=ValEx("0x1414")
Print("ValEx(""0x1414"")="+str(i))
```

ValEx - Convert Decimal, Hexadecimal, or Octal String to Integer

Valid on UNIX and Windows

The ValEx function converts the input string, which specifies a decimal, a hexadecimal, or an octal number, to an integer.

Function format:

```
ValEx(str as String) as Integer
```

str

The string specifies a decimal, a hexadecimal, or an octal number, and can include a sign. The number specified is converted to an integer.

The function returns the integer value of the input string. If the string cannot be converted to a number, the function returns 0.

Example:

```
Dim i As Integer
i = Val("2735")
Print("Val(""2735"")"+Str(i))
i=ValEx("2735")
Print("ValEx(2735")
Print("ValEx(""2735"")="+str(i))
i=ValEx("01414")
Print("ValEx(""01414"")="+str(i))
i=ValEx("0x1414")
Print("ValEx(""0x1414"")="+str(i))
```

Functions for Splitting Strings

To help parse information read from files or other sources, you can split strings into separate tokens or substrings using the following DMScript functions. A token is a substring separated from others by a delimiter, usually a space. You can quote tokens with single or double quotes.

- SetTokeniserInput
- GetToken

SetTokenizerInput

The SetTokenizerInput function initializes the tokenizer with a string and defines the delimiters for splitting the string. This string and delimiters are valid for the rest of the script until you call the function with a different string or delimiter.

Function format:

```
SetTokenizerInput(str as string, delimiters as string)
```

Input Parameters

This function has the following input parameters:

str

Specifies the input string that you want to split.

Note: You can specify only one input string at a time.

delimiters

Specifies the set of delimiters for splitting the string. By default, space characters are used as delimiters, but you can specify any set of characters. For example, “.,/” tokenizes a list of substrings separated by a dot, comma or slash.

Return Values

None

GetToken

The GetToken function retrieves the next string token from the current input string set by SetTokenizerInput.

Function format:

```
GetToken(token as string)as boolean
```

Example:

```
SetTokenizerInput("This is a test program", " ")
while GetToken(token)
  print token
wend
```

Input Parameters

This function has the following input parameters:

token

Specifies a string variable that holds the next token returned by the function.

Return Values

Returns True if it returned the next string token. It returns False if no more tokens are available to retrieve from the current input string.

Output

```
This is a test program.
```

User-defined Functions

This script demonstrates how to use functions in AM scripts.

```
REM
REM This script demonstrates how to use functions in AM
REM scripts
REM

'*********************************************************
'*
'* This function adds two integers and return the sum
'*
'*********************************************************
Function sum( a as Integer, b as Integer) as integer
 DIM result AS Integer

 result = a + b

 'Return the result as the function value
 sum = result
End Function 'sum

'*********************************************************
'************************--Main --************************
'*********************************************************
'This is the actual program
Print str(Sum(3,4))                    '7
Print str(Sum(234,567))                          '801
```

Example: Using DLLs from Script Function

This script demonstrates how to use functions in asset management scripts.

```
Rem
Rem This example sets the wallpaper on a Windows NT or Windows 95/98,
Rem using a system DLL call.
Rem
#if Win32
  '32-Bit declaration of the function
 Function SystemParametersInfo Lib "USER32.DLL" Alias "SystemParametersInfoA" ( uiAction as Word,uiParam
as Word,pvParam as String, fWinIni as Word) as Boolean
#EndIf

'Define Flags
Dim SPI_SETDESKWALLPAPER as integer
Dim SPIF_UPDATEINIFILE as integer
Dim SPIF_SENDWININICHANGE as integer
'Set system flags
SPI_SETDESKWALLPAPER = 20
SPIF_UPDATEINIFILE = 0x0001
SPIF_SENDWININICHANGE = 0x0002

'Call the system function
SystemParametersInfo(SPI_SETDESKWALLPAPER,0,"c:\winnt\winnt.bmp", SPIF_UPDATEINIFILE OR
SPIF_SENDWININICHANGE)
```