NetOp Scripting API

NFMScrpt.ocx is installed in your Windows system32 directory when you install a NetOp Guest. It allows you to access the Guest's scripting capabilities from any programming or scripting tool, which supports ActiveX automation. One commonly used tool is Microsoft Visual Basic (VB). The ocx is tested with VB, and examples in this help text will be written mostly in VB. An example of a VBscript using a excerpt of the commands available is

```
Rc = Script.Initialize()
Rc = Script.Call("MyDesktop")
Rc = Script.IncludeSubdirectories(True)
Rc = Script.Synchronize("c:\MyDocuments\*.*", "c:\MyDocuments\*.*")
Rc = Script.Hangup()
Rc = Script.Uninitialize()
```

Scripts as simple as this are more easily created and executed with the Script editor in the NetOp guest program. Say however, you wish to retry all or parts of your operations repeatedly until they have all succeeded, you must make a more complex algorithm, which this editor is not designed for. With the NFMscript ocx you can improve the above script to for example:

```
        Rc = Script.Initialize()
CallAgain:
        Rc = Script.Call("MyDesktop")
        Rc = Script.IncludeSubdirectories(True)
        RcSync = Script.Synchronize("c:\MyDocuments\*.*", "c:\MyDocuments\*.*")
        Rc = Script.Hangup()
        if (RcSync<>0) Then
                WriteLog ("Failed. Trying again in 30 seconds")
                WaitSeconds(30)
                GoTo CallAgain:
        End If
        Rc = Script.Uninitialize()
```

NetOp Scripting API - Creation and Deletion

An NFMscript object is created and eventually destroyed with the means of the programming tool. With VB, you can use the visual way by right clicking the object toolbar (the one on the left side), and choose Components. A dialog with all available OCXs appears. Check the box with "Danware NetOp File Manager Script", and press Ok. A script icon will be added to your toolbar. Click this icon, then click the location in the Form where you wish the NFM script object placed, and drag it out. The default visual representation is a treeview showing commands as they execute, so even though the control current shows up blank, it may be an idea to give it a reasonable size.

Script.ClearLog() can be used to clear the tree view log window. If you do not wish any visual feedback, you can make the script invisible. You can also choose another reporting mode than ReportLog().

```
Set    Script.Visible = False
Rc = Script.ReportSilent()
Rc = Script.ReportStatus()
Rc = Script.ReportLog()
```

The OCX can handle any number of simultaneous NFMscript objects, but the NetOp Guest version 6.0 will limit you to maximum 10 active objects at a time. The 11th and all further objects can be created, but will return error codes from all methods.

NetOp Script API - StartGuest, FreeGuest, Initialize and Uninitialize

The NFMscrpt.ocx is only another way of wrapping up the NetOp guest. Therefore, the NetOp guest program has to be running when the ocx executes. The simplest way is to start it manually before starting the program or script you are writing using Nfmscrpt.ocx

You may however want to hide the NetOp guest program, and consider it an invisible service which happens to need to run with your application. If you wish that you can call the StartGuest function
In VB you would typically do that in the Form_Load() function for your initial form

```
Sub Form_Load()
Dim Rc As Long
Again:
        Rc = Script.StartGuest(True)
        if (Rc < -12 Or Rc > -11) Then
                MsgBox("Please Exit NetOp Host")
                GoTo Again
        End If
End
```

If NetOp is installed and is working properly, the most likely reason for not being able to start the guest program is that the host is running. You must manually stop the host. When the guest has started, you can send commands to it from any NFMscript object you have created. The first command any object should send is the Initialize command, which creates connection between the object and the guest. This will typically happen as a reaction on the click of a button.

```
Sub Button_Click()
        Rc = Script.Initialize()
        if (Rc <> 0) Then
                MsgBox("No connect. Is NetOp Guest Running?")
                GoTo EndButtonClick
        End If

        '< ... do your stuff ...>

        Rc = Script.Uninitialize()
EndButtonClick:
End
```

One reason the Initialize might fail and return non zero, might be that the guest program could not start. It is good practise to call Uninitialize when you are returning from your subroutine. This way you will free the connection to the guest to be used for others. If you forget to Uninitialize, it will be done implicit for you if you call Initialize again, but you will be blocking 1 out of 10 connections to your guest in the meanwhile.

Uninitialize returns 0 on success and a non-zero code on error. You need not take any speficic action if an error is returned.

When your application exits, it is good practice to call FreeGuest(), which will do all needed clean up. Your program will work ok without a call to FreeGuest, but you will be relying on the program exit to clean everything up.

Note: If you are writing a script for browser use (i.e. Internet Explorer), do not call FreeGuest(), as you are not the one to decide when Internet Explorer exits.

```
Sub StopButton_Click()
```

```
        Rc = Script.FreeGuest()
        Stop
End
```

SUMMARY

StartGuest() may be called once at program start, no matter how many NFMscript objects you wish to create. FreeGuest should be called on exit.

Initialize() must be called before any other command. The one exception is StartGuest().

After Uninitialize(), no other commands but FreeGuest() will succeed until the next Initialize().

You can have any number of Initialize() ... Uninitialize() sessions on the same object.

NetOp Script API - Call and Hangup

The next thing you have to do is to call a NetOp host program running on another computer. The Call() command will establish this connection for you. If it fails, it will return a non-zero error code. If it succeeds, it will return 0.

The argument to Call() is a String which is the name of the NetOp phonebook (.dwc) file. In this file is stored the name of a computer and the parameters for how to connect to it. The phonebook files are the ones shown in the NetOp guest programs Phonebook tab control. Say you have a phonebook file named "Venus.dwc":

```
Sub Button_Click()
        Rc = Script.Initialize()
        Rc = Script.Call("Venus")
        if (Rc <> 0) Then
                MsgBox("Venus not responding")
                GoTo EndButtonClick
        End If

        '< ... do your stuff ...>

        Rc = Script.Hangup()
        Rc = Script.Uninitialize()
EndButtonClick:
End
```

It is good practise to call Hangup() before you make your next Call(). If you happen to make a new Call() before Hangup() on the first one, it will be hung up automatically. One good reason not to omit calling Hangup is to save money on your telephone bill. You can make as many Call()'s and Hangup()'s you might wish on the same object.

Please be aware that the argument to Call() is NOT the name of the computer you wish to call. It is the name of a phonebook file. As such files often reside in the NetOp phonebook directory, you need not specify a path if you have the file there. As the NetOp default for phonebook filename extension is ".dwc", you need neither pass that, so the three calls do the same, but the two last are independent of where NetOp is installed.

```
Script.Call("C:\program files\netop remote control\phbook\venus.dwc")
Script.Call("venus.dwc")
Script.Call("venus")

Script.Call("*")
```

The fourth call does not know which phonebook file it wants to use. the "*" parameter will cause a file selection box to pop up, where the end user can select a .dwc file in the phonebook directory. If you wish to make more advanced programs, you can traverse the phonebook directory and construct a dynamic list of phonebook files.

TRAVERSING THE PHONEBOOK

If you wish a control which makes the phonebook files available as other than the independent popup file selection box made with Script.Call("*"), you can traverse the phonebook directory like for example below, where a combo box is used

```
        Sub Combo1_Dropdown()
        Dim More As Boolean
                More = Script.PhonebookSetFirst()
                Do While (More)
```

```
                Combo1.Add(Script.PhonebookGetName())
                More = Script.PhonebookGetName()
        Loop
End Sub

Sub Combo1_Click()
        Script.Call(Combo1.Value)
        ...
        Script.Hangup()
End Sub
```

If you wish to traverse only a subset of all your phonebook connections, place the ones you wish to expose in a subfolder named "offices", for example using the Phonebook tab control in the NetOp guest program, then use

```
Script.PhonebookSetSubfolderFirst("offices")
```

Summary

Call() must be called to connect to a host. After a successful Call() you can call other commands. Do Call("*") to make a computer independent script.

When done with the host, call Hangup(). After a Hangup(), no commands which need host access will succeed.

You can have any number of Call() ... Hangup() connections on the same object.

NetOp Script API - Transferring Files

After a Call() and before a Hangup(), you can call the file transfer commands which are

Script.CopyFromHost (RemoteFileFilter, LocalDirectory)
Script.CopyToHost (LocatFileFilter, RemoteDirectory)
Script.CloneFromHost (RemoteDirectory, LocalDirectory)
Script.CloneToHost (LocalDirectory, RemoteDirectory)
Script.Synchronize (LocalDirectory, RemoteDirectory)
Script.SynchronizeOneway (LocalDirectory, RemoteDirectory, Direction)

Remote indicates files on the remote computer where the NetOp host program runs, Local is the machine where your NFMscript application and the NetOp guest run.

File filters must be legal Windows file filters like "C:\winnt*.exe". The name of one single file like "C:\config.sys" is also a legal file filter. Blanks are allowed in names. The functionality of these commands are described in the Chapter on scripting from inside NetOp.

The dialogs of NetOp are not shown during the execution of the commands unless the command need it's end user to take a decision, for example if a file should be overwritten or not. But if you call for example CopyToHost on a very large file via a slow telephone line, your application is not locked. In your script program

- All events are still processed so any button can be pressed
- Progress of commands can be caught and monitored
- Cancelling commands is build-in, and can even be customized

IMPORTANT NOTICE

The methods in an Nfmscript object are not reentrant. In order to keep your application alive and responsive, all messages are processed while the method waits for NetOp to finish processing the method. This makes it possible for you to call the same method again while the first call your made has not returned yet. Such a call will not work correctly, but return a busy code. It is your application's responsibility to ensure that methods in the Nfmscript objects are not reentered into. One very useful exception to this rule is the three cancel methods.

CANCEL

If you have chosen to have your NFMscript visible in your application, your end user can press the escape key in the script log window. This fires the internal OnCancel() event. The built-in action on that event is that a messagebox pops up with a choice of four actions

- **Continue (Action 0)**
- **Cancel Command (Action 1)**
- **Cancel Call (Action 2)**
- **Cancel Script (Action 3)**

Continue, will cause the script to continue as has nothing happened. In fact, the NetOp guest is never notified. All three other NFMscript cancel replies will send a Cancel() command to NetOp. NetOp will in turn, as promptly as possible cancel the last command it was sent from your script, and that script function will return with an error. What will happen next is different for each of the three cancel replies.

Choosing Cancel Command will cause the next script command to be issued to NetOp. Only one single script command was stopped. Cancel Command is intended for use when for example one large irrelevant file blocks a useful transfer of many files.

Cancel Call will cause all further script commands to be ignored until the next Hangup command. All commands from the current one and till next Hangup will simply return successfully without doing anything. Cancel Call addresses the situation where you for example picked the wrong computer to connect to.

Cancel Script works the same way, but until the next Uninitialize command. It is intended for use when you want to stop everything and evaluate what to do next.

If you wish to have your own interface to cancelling, you can use the three equivalent cancel commands from the script interface. Since all events are still being processed during the excution of a command like CopyToHost(), all buttons will respond at any time. From your own cancel button, call

```
Script.CancelCommand() or
Script.CancelCall() or
Script.CancelScript()
```

for example like this, if you designed a button named CancelButton:

```
Sub CancelButton_Click()
        Script.CancelCall()
End Sub
```

If you wish to use the internal cancel event, but construct your own actions on that event, fill in the OnCancel() event which the OCX will fire on your script application before putting up its message box.
You can for example do like this to make the user dialog less complex by allowing only CancelScript:

```
Private Sub Script_OnCancel(Action As Long)
        rc = MsgBox("Cancel ?", vbYesNo)
        If rc = vbYes Then Action = 3
        If rc = vbNo Then Action = 0
End Sub
```

In the parameter Action you return 0 for continue, 1 for cancel command, 2 for cancel call and 3 for cancel script. Action will arrive to you with a value of -1. If you do not change that value, the built in message box above will pop up, otherwise not.

ADDING AN OPTION DIALOG

In parallel with OnCancel(), you will find OnRbuttonDown(). A difference is that this event has no default action. It only does what you program. The parameter is to there allow future extensions. For forwards compatibility, return a zero for no action.

```
Private Sub Script_OnRbuttonDown(Action As Long)
        rc = MsgBox("Include Subdirectories", vbYesNo)
        If rc = vbYes Then Script.SetIncludeSubdir(True)
        If rc = vbNo Then Script.SetIncludeSubdir(False)
        Action = 0
End Sub
```

MONITORING PROGRESS

You can at any time query the progress of a script command. It is however your application’s responsibility to find a suitable place in your code to do it from. The NFMscript exposes the function

Script.GetProgress()

which returns a percentage between 0 and 100. To use this from VB, instance a timer and a progress bar. You can for example get the progress bar from one of the Microsoft common controls ocx's.

```
Sub Button_Click()
        rc = Script.Call(..)
        Timer1.Interval = 500
        rc = CopyToHost(....)
        Timer1.interval = 0
        Script.Hangup()
End Sub

Sub Timer1_Timer()
        ProgressBar1.Value = Script.GetProgress()
EndSub
```

SETTINGS

The NetOp scripting has many parameters to the file transferring commands. All these have been made available as methods named Set<NameOfItem>() in the OCX. They are

SetOverwriteReadonly(BOOL YesNo)
SetOverwriteHidden(BOOL YesNo)
SetOverwriteSystem(BOOL YesNo)
SetOverwriteExisting(BOOL YesNo)
SetRetriesOnTransferError(long Retries)
SetRetriesOnConnectError(long Retries)
SetDeltaFileTransfer(BOOL YesNo)
SetCrashRecovery(BOOL YesNo)
SetCompression(long Level)
SetConnected(BOOL conn)
SetIncludeEmptyDir(BOOL YesNo)
SetIncludeSubDir(BOOL YesNo)
SetIncludeHiddenAndSystem(BOOL YesNo)
SetIncludeOnlyNewer(BOOL YesNo, DATE DateTime)
SetIncludeOnlyExisting(BOOL YesNo)
...

You may ask why these are methods and not properties, since all they seem to do is set the value of a variable. The reason is that some of them needed to be implemented as sending real commands to NetOp, while others just set a value to be used as an option to another command. For consistency, all settings are implemented as methods.

EXECUTE

Many methods in the Nfmscrpt.ocx correspond to a command in the NetOp script command language, which is the syntax you see in the NetOp guest's script editor dialog and also in the OCX's log window. If you wish, you can send commands directly in that command language using

Rc = Script.Execute(String Command);

The purpose of this OCX is however to relieve you of the burden of a lot of string formatting and event handling, so this entry is only published as an extra service for unforeseen circumstances.

NetOp Script API - Examples

In the NetOp installation directory, you will find a file named Examples.zip. Unzip this file to get the source code and executables for the examples. You will find

HELLO WORLD SCRIPT

HelloWorldScript.exe is the simplest possible example. When you press the start button, it will copy a file to a host computer. The Visual Basic project HelloWorldScript.vbp is included in the installation

```
Private Sub Command1_Click()
    Dim Rc As Long
    Rc = HelloScript.Initialize
    Rc = HelloScript.Call("*")
    ' Move some arbitrary file across. This one is always there
    Rc = HelloScript.CopyToHost(HelloScript.GetInstallDir() + "\netop.fac", "c:\*.*")
    Rc = HelloScript.Hangup
    Rc = HelloScript.Uninitialize
End Sub

Private Sub ExitButton_Click()
    HelloScript.FreeGuest
    Stop
End Sub

Private Sub Form_Load()
    HelloScript.StartGuest (True)
End Sub
```

VISIT ALL HOSTS SCRIPT

This example is a bit more feature rich. In the beginning we declare a logical variable and we start the NetOp guest when the program starts up. Next we cycle through the available phonebook files in the phonebook root directory and write their names in the log. We are namely intending to visit all these hosts one by one.

```
Dim More As Boolean

Private Sub Form_Load()
    Script.StartGuest True
    More = Script.PhonebookSetFirst
    Do While More
        Script.WriteLog "Will visit " + Script.PhonebookGetFilename
        More = Script.PhonebookSetNext
    Loop
End Sub
```

There is a button labelled “Start Visit”. When that is clicked, we show a dialog in which we will show what we are doing with that host while doing a CopyToHost() operation. When we are finished we stop the dialog and hide it.

```
Private Sub StartButton_Click()
    StartButton.Enabled = False
    StopButton.Enabled = True
```

```
    Script.Initialize
    More = Script.PhonebookSetFirst
    Do While More
        rc = Script.Call(Script.PhonebookGetFilename)
        VisitDialog.Show
        Script.CopyToHost Script.GetInstallDir + "\netop.fac", "c:\*.*"
        VisitDialog.Animation1.AutoPlay = False
        VisitDialog.Timer1.Interval = 0
        Script.Hangup
        VisitDialog.Hide
        More = Script.PhonebookSetNext
    Loop
    StopButton.Enabled = False
    StartButton.Enabled = True
    Script.Uninitialize
End Sub
```

The dialog shows the .avi file with the filecopy animation which also explorer does. The dialog has a timer which updates a progress bar.

```
Private Sub Form_Load()
    Caption = VisitForm.Script.PhonebookGetFilename
    Timer1.Interval = 100
    Animation1.Open "d:\netop\v60\filecopy.avi"
    Animation1.AutoPlay = True
End Sub

Private Sub CancelButton_Click()
    VisitForm.Script.CancelCall
    Hide
End Sub

Private Sub Timer1_Timer()
    ProgressBar1.Value = VisitForm.Script.GetProgress
    ProgressBar1.Refresh
End Sub
```

KEEP SYNCHRONIZED SCRIPT

This is example showing timing and repetition using the Wait…() functions. At startup, start the guest ans set the initial parameters for the interface and the internal variables:

```
Dim Rc As Long
Dim TryAgain As Boolean

Private Sub Form_Load()
    Script.StartGuest (True)
    TryAgain = True
    StartTime.Value = Now
    ' StartDate.Value = Today
End Sub
```

I tried also doing StartDate.Value = Today, which would be elegant, but my version of Visual Basic insists that Today is year 1899, so I commented it out. The WaitUntil() function holds execution until the date and time antered

into the Microsoft DTPicker controls StartDate and StartTime. Call("*") leaves it up to the end user to pick a phonebook file in a FileDialog, then Synchronize() synchronizes the contents of two directories. If the interface's checkbox is checked, the program will try repeat the Call() and Synchronize() periodically until you actively stop it. While inactive, the program will hide itself.

```
Private Sub StartButton_Click()
    Rc = Script.Initialize
    Rc = Script.WaitUntil(StartDate.Value, StartTime.Value)
Again:
    Rc = Script.Call("*")
    If (Rc <> 0) Then GoTo Done
    Rc = Script.Synchronize("C:\reports\*.*", "c:\reports\*.*")
    If (Rc <> 0) Then MsgBox ("This example assumes a directory C:\REPORTS")
    Rc = Script.Hangup
    If (Repeat.Value = Checked And TryAgain) Then
        If (MsgBox("Now sleep: " + CStr(Interval.Value), vbOKCancel) _
            = vbCancel) Then GoTo Done
        KeepInSyncForm.Hide
        Script.Wait (Interval.Value)
        KeepInSyncForm.Show
        GoTo Again
    End If
Done:
    Rc = Script.Uninitialize
End Sub
```

The button labelled Stop will cancel the repeating cycles

```
Private Sub StopButton_Click()
    Script.CancelScript
    TryAgain = False
End Sub
```

The button labelled Clear will clear the log. This can be useful if it becomes very long.

```
Private Sub ClearButton_Click()
    Script.ClearLog
    Script.WriteLog ("Ready")
End Sub
```

The Exit button will free the guest and stop the program.

```
Private Sub ExitButton_Click()
    Script.FreeGuest
    Stop
End Sub
```

If you hold down the right mouse button, you can clear the log.

```
Private Sub Script_OnRbuttonDown(Action As Long)
    If (MsgBox("Clear Log?", vbYesNo) = vbYes) Then
        ClearButton_Click
        Action = 0
    End If
End Sub
```

NetOp Script API - Reference

All Nfmscript methods which return a Long, return zero for success.

Call (Filename As String) As Long
    Filename
CancelCall () As Long
CancelCommand () As Long
CancelScript () As Long
ClearLog () As Long
CloneFromHost (RemoteDir As String, LocalDir As String) As Long
CloneToHost (LocalDir As String, RemoteDir As String) As Long
CopyFromHost (RemoteFilter As String, LocalDir As String) As Long
CopyToHost (LocalFilter As String, RemoteDir As String) As Long

DirGetName () As String
DirSetFirst (Directory s String) As Boolean
DirSetNext () As Boolean
DriveGetName () As String
DriveSetFirst () As Boolean
DriveSetNext () As Boolean

Execute (Command as String) As Long

FileGetAccessed () As Date
FileGetArchive () As Boolean
FileGetCreated () As Date
FileGetHidden () As Boolean
FileGetModified () As Date
FileGetName () As Date
FileGetReadonly () As Boolean
FileGetSize () As Long
FileGetSystem () As Boolean
FileSetFirst (FileFilter As String) As Boolean
FileSetNext () As Boolean
FreeGuest () As Long

GetInstallDir () As String
GetProgress () As Long
GetPhonebookDir () As String

Hangup () As Long

Initialize () As Long

PhonebookGetFilename () As String
PhonebookSetFirst () As Boolean
PhonebookSetNext () As Boolean
PhonebookSetSubfolderFirst (Folder As String) As Boolean

RunLocal (Command As String) As Long
RunRemote (Command As String) As Long

SetCompression (Level As Long) As Long
SetCrashRecovery (YesNo As Boolean) As Long
SetDeltaFileTransfer (YesNo As Boolean) As Long

SetIncludeEmptyDir (YesNo As Boolean) As Long
SetIncludeHiddenAndSystem (YesNo As Boolean) As Long
SetIncludeOnlyExisting (YesNo As Boolean) As Long
SetIncludeOnlyNewer (YesNo As Boolean, Date As Date) As Long
SetIncludeSubDir (YesNo As Boolean) As Long
SetOverwriteReadonly (YesNo As Boolean) As Long
SetOverwriteHidden (YesNo As Boolean) As Long
SetOverwriteSystem (YesNo As Boolean) As Long
SetOverwriteExisting (YesNo As Boolean) As Long
SetReportSilent () As None
SetReportStatus () As None
SetReportLog () As None
SetRetriesOnTransferError (Retries As Long) As Long
StartGuest (Minimized As Boolean) As Long
Synchronize (LocalDir As String, RemoteDir As String) As Long
SynchronizeOneWay (SourceDir As String, TargetDir As String, ToHost As Boolean) As Long

Uninitialize () As Long

Wait (Period As Date) As Long
WaitSeconds (Period As Long) As Long
WaitUntil (Date As Date, Time As Date) As Long
WaitUntilAnyDay (Time As Date) As Long
WriteLog (Text As String) As Long

Call (Filename As String) As Long

Call a phonebook entry. See also Hangup() and CancelCall(). If Initialize() was not called, it will be called implicitly. That will in turn call StartGuest() if the guest is not already running. If another Call() is currently active, it will be hung up. If you wish two simultaneous Call()'s you must use two Nfmscript objects.

Filename:

The phonebook filename. If it has no extension, ".dwc" will be added. If it has no path, the NetOp phonebook directory will be prepended. The NetOp.ini PHONEBOOKPATH and DATAPATH settings are respected.

CancelCall () As Long

Cancel the Call() which is currently active. Typically called asynchronously from a separate button. The current method (for example CopyFromHost) will be cancelled, and return an error code. All following methods will return immediately with no error, until your program executes the next Hangup() or Call() method.

CancelCommand () As Long

Cancel the method call which is currently active. Typically called asynchronously from a separate button. The current method (for example CopyFromHost) will be cancelled, and return an error code. All following methods will execute as if nothing had happened.

CancelScript () As Long

Cancel the Call() which is currently active. Typically called asynchronously from a separate button. The current method (for example CopyFromHost) will be cancelled, and return an error code. All following methods will return immediately with no error, until your program executes the next Uninitialize() or Initialize() method.

ClearLog () As Long

Clears the script object's log window.

CloneFromHost (RemoteDir As String, LocalDir As String) As Long

Clones the directory RemoteDir to the LocalDir directory. A Call() must be open to the computer with the RemoteDir.

RemoteDir
A directory on the remote computer where the NetOp host runs. Must end with "*.*".

LocalDir
A directory on you local computer where the NetOp guest runs. Must end with "*.*".

CloneToHost (LocalDir As String, RemoteDir As String) As Long

Clones the directory LocalDir to the RemoteDir directory. A Call() must be open to the computer with the RemoteDir.

LocalDir
A directory on your local computer where the NetOp guest runs. Must end with "*.*".

RemoteDir
A directory on the remote computer where the NetOp host runs. Must end with "*.*".

CopyFromHost (RemoteFilter As String, LocalDir As String) As Long

Clones the files matching RemoteFilter to the LocalDir directory. A Call() must be open to the computer with the RemoteFilter.

RemoteFilter
A valid file filter on the remote computer where the NetOp host runs. An example could de "C:\DATA*.XLS"

LocalDir
A directory your local computer where the NetOp guest runs. Must end with "*.*".

CopyToHost (LocalFilter As String, RemoteDir As String) As Long

Clones the files matching LocalFilter to the RemoteDir directory. A Call() must be open to the computer with the RemoteDir.

LocalFilter
A valid file filter on your local computer where the NetOp guest runs. An example could de "C:\DATA*.XLS"

RemoteDir
A directory on the remote computer where the NetOp host runs. Must end with "*.*".

DirGetName () As String

Returns the name of the current subdirectory from DirSetFirst/Next().

DirSetFirst (Directory s String) As Boolean

Initializes the directory search entries, so next call to DirGetName() will return the name of the first subdirectory of "Directory" on the remote computer. You must have an open Call() to that computer. If there are no such subdirectories, the return value is False. On success, the return value is True.

Directory

A directory on the currently Call()'ed remote computer.

DirSetNext () As Boolean

Advances to the next directory search entry, so next call to DirGetName() will return the name of the next subdirectory. If there are no more subdirectories, the return value is False. On success, the return value is True.

DriveGetName () As String

Returns the name of the current disk drive from DriveSetFirst/Next().

DriveSetFirst () As Boolean

Initializes the disk drive entries, so next call to DriveGetName() will return the name of the first disk drive on the remote computer file, you currently have made a Call() to. If there are no disk drives, the return value is False. On success, the return value is True.

DriveSetNext () As Boolean

Advances to the next disk drive entry, so next call to DriveGetName() will return the name of the next disk drive. If there are no more drives, the return value is False. On success, the return value is True.

Execute (Command as String) As Long

Execute a script editor command. The format of these command resemble the Nfmscript methods, and are documented in The NetOp User's Manual.

Command:
The command to execute

FileGetAccessed () As Date

Returns the last access date for the file selected with FileGetFirst/Next()

FileGetArchive () As Boolean

Returns the archive flag for the file selected with FileGetFirst/Next()

FileGetCreated () As Date

Returns the create date for the file selected with FileGetFirst/Next()

FileGetHidden () As Boolean

Returns the hidden flag for the file selected with FileGetFirst/Next()

FileGetModified () As Date

Returns the modified date for the file selected with FileGetFirst/Next()

FileGetName () As Date

Returns the name of the file selected with FileGetFirst/Next()

FileGetReadonly () As Boolean

Returns the readonly flag for the file selected with FileGetFirst/Next()

FileGetSize () As Long

Returns the size of the file selected with FileGetFirst/Next(). If the size is above 2GB, -1 will be returned.

FileGetSystem () As Boolean

Returns the system flag for the file selected with FileGetFirst/Next()

FileSetFirst (FileFilter As String) As Boolean

Initializes the file entries, so next call to FileGet…() will return a property of the first file on a remote computer matching the given file filter. If there are no entries, the return value is False. On success, the return value is True. There must be an open Call() to the remote computer.

FileFilter
A legal file filter on the remote computer like for example "C:*.*"

FileSetNext () As Boolean

Advances to the next file entry, so next call to FileGet… () will return the name of the next remote file. If there are no more files, the return value is False. On success, the return value is True

FreeGuest () As Long

Frees connection to NetOp guest DLLs and does other clean up. Not mandatory, but it is good practice to call this before your application exits.

GetInstallDir () As String

Returns the NetOp install directory on your local computer where the NetOp guest program runs.

GetProgress () As Long

Get the progress of the current method. Typically only useful with Copy, Clone and Synchronize methods. Returns the percentage 0-100 where 100 means done. Useful if you place it in a timer and feed the result into a progress bar.

GetPhonebookDir () As String

Returns the phonebook directory. The NetOp.ini PHONEBOOKPATH and DATAPATH settings are respected.

Hangup () As Long

Hang the current Call() up.

Initialize () As Long

Initializes a session with a NetOp guest. Check the return code to be zero before calling other methods. See also Uninitialize(). If the NetOp guest is not already running, StartGuest() will be called implicitly.

PhonebookGetFilename () As String

Returns the name of the current phonebook file. If there are none, the string returned is "No Phonebook Entries or Error"

PhonebookSetFirst () As Boolean

Initializes the phonebook entries, so next call to PhonebookGetFilename() will return the name of the first phonebook file. If there are no entries, the return value is False. On success, the return value is True.

PhonebookSetNext () As Boolean

Advances to the next phonebook entry, so next call to PhonebookGetFilename() will return the name of the next phonebook file. If there are no more files, the return value is False. On success, the return value is True. Can be used with both PhonebookSetFirst() and PhonebookSetSubfolderFirst().

PhonebookSetSubfolderFirst (Folder As String) As Boolean

Initializes the phonebook entries, so next call to PhonebookGetFilename() will return the name of the first phonebook file is a specific subdirectory of the phonebook directory. If there are no entries, the return value is False. On success, the return value is True.

RunLocal (Command As String) As Long

Runs an operating system executable file with parameters on your local computer.

Command

The name of a bat, com or exe file. If you wish to use shell commands, use must give the name of the shell executable. For NT and Win95 this is "cmd.exe", so you can use "cmd /c dir c:*.*" or "cmd /k rename autoexec.bat autoexec.old".

RunRemote (Command As String) As Long

Runs an operating system executable file with parameters on a remote computer. A Call() must be open to that computer. Please note that the outcome this is dependent of the setup of the remote computer environment, and 100% independent of your local computer.

Command

The name of a bat, com or exe file. If you wish to use shell commands, use must give the name of the shell executable. For NT and Win95 this is "cmd.exe", so you can use "cmd /c dir c:*.*" or "cmd /k rename autoexec.bat autoexec.old".

SetCompression (Level As Long) As Long

Set the compression level.

Level

An integer number. 0 means no compression, >0 means compression.

SetCrashRecovery (YesNo As Boolean) As Long

Instructs NetOp whether or not to try apply crash recovery. If a call() is interrupted, a partial file can be kept on the target disk. Only useful if delta file transfer is on, so this method will implicitly set delta file transfer to True.

YesNo

If True, partial files will be kept on the target disk, and delta file tranfer will be set, so the valid part need not be retransmitted next time you come back. If False, partial files will be cleaned up automatically if you loose your connection.

SetDeltaFileTransfer (YesNo As Boolean) As Long

Instructs NetOp whether or not to try apply Delta File Transfer, the method to try minimize the amount of data transferred unnecessarily. This state is also set by SetCrashRecovery(True), but not cleared by SetCrashRecovery(False).

YesNo

If True, delta file transfer will be applied when applicable. If False, all file transfers will unconditionally transfer all bytes in all files.

SetIncludeEmptyDir (YesNo As Boolean) As Long

Instructs NetOp whether or not to include empty directories in file transfer operations.

YesNo

If True, empty directories are included. If False, they are not included

SetIncludeHiddenAndSystem (YesNo As Boolean) As Long

Instructs NetOp whether or not to include hidden and system files in file transfer operations.

YesNo

If True, hidden and system files are included. If False, they are not included.

SetIncludeOnlyExisting (YesNo As Boolean) As Long

Instructs NetOp whether or not to include only files which already exist with the same name on the target computer in file transfer operations.

YesNo

If True, only files which already exist with the same name on the target computer are transferred; no new files will be created. If False, all files are transferred whether they exist in advance or not.

SetIncludeOnlyNewer (YesNo As Boolean, Date As Date) As Long

Allows you to set a limit to how old files you wish to include in file transfer operations.

YesNo

If True, only files which are newer than Date are transferred. If False, all files are transferred no matter what date they were last modified.

Date

Files with a modify date older than this will be excluded if YesNo is True.

SetIncludeSubDir (YesNo As Boolean) As Long

Instructs NetOp whether or not to include subdirectories of the directories/file filters given as source in file transfer operations.

YesNo

SetOverwriteReadonly (YesNo As Boolean) As Long

Set the action you wish when trying to overwrite readonly files

YesNo

If True, readonly files will be overwritten without warnings. If False, readonly files will cause a prompt in a dialog.

SetOverwriteHidden (YesNo As Boolean) As Long

Set the action you wish when trying to overwrite hidden files

YesNo

If True, hidden files will be overwritten without warnings. If False, hidden files will cause a prompt in a dialog.

SetOverwriteSystem (YesNo As Boolean) As Long

Set the action you wish when trying to overwrite system files

YesNo

If True, system files will be overwritten without warnings. If False, system files will cause a prompt in a dialog.

SetOverwriteExisting (YesNo As Boolean) As Long

Set the action you wish when trying to overwrite existing files

YesNo

If True, existing files will be overwritten without warnings. If False, existing files will cause a prompt in a dialog.

SetReportSilent () As None

Disable the logging of events in the object's log window

SetReportLog () As None

Make the logging of events in the object's log window be the default treeview representation.

SetRetriesOnTransferError (Retries As Long) As Long

Set the number of times you wish the file transfer methods to automatically retry an operation before returning

Retries

An integer number between 0 and 9 inclusive.

SetRetriesOnConnectError (Retries As Long) As Long

Set the number of times you wish the file call method to automatically retry making the connection before returning

Retries
An integer number between 0 and 9 inclusive.

StartGuest (Minimized As Boolean) As Long

Starts the NetOp guest executable. If it is already started, StartGuest() will just return. If the NetOp host is running, StartGuest() will return with an error code. If StartGuest() succeeds, it will return –11 or –12.

Minimized

If True, the guest will be attempted started up minimized.

Synchronize (LocalDir As String, RemoteDir As String) As Long

Synchronizes two directories. A Call() must be open to the computer with the RemoteDir.

LocalDir
A directory on your local computer where the NetOp guest runs. Must end with "*.*".

RemoteDir
A directory on the remote computer where the NetOp host runs. Must end with "*.*".

SynchronizeOneWay (SourceDir As String, TargetDir As String, ToHost As Boolean) As Long

Synchronizes two directories, but moves files one way only. A Call() must be open to the remote computer.

SourceDir
the directory where the files origin from. It can be local or remote depending on ToGuest. Must end with "*.*".

TargetDir
The target directory. It can be local or remote depending on ToGuest. Must end with "*.*".

ToHost
If True, files are only transferred from guest to host. If False, it is reverse.

Uninitialize () As Long

Uninitializes a session with a NetOp guest. Initialize() must be made again before calling other methods. Uninitialize is not mandatory, but a good practice.

Wait (Period As Date) As Long

Waits a period, then returns

Period

The interval you wish the method to wait before returning. If you have only a number of seconds the WaitSeconds() function is easier because it does not require a Date variable. Not that if you have AM-PM time representation, an interface showing 12:00:01 AM will cause a wait of 1 second, not 12 hours and 1 second.

WaitSeconds (Period As Long) As Long

Waits a number of seconds, then returns

Period
The interval you wish the method to wait before returning.

WaitUntil (Date As Date, Time As Date) As Long

Waits until a given local time and date, then returns. For easier use with the MS DTPicker object, this method has two parameters. You can have two DTPickers, one for date and one for time.

Date
The date you wish the method to wait until before returning. If this vaiable has a Time part, it will be ignored.

Time
The time on the above date when the method will return. If this variable has a Date part it is ignored.

WaitUntilAnyDay (Time As Date) As Long

Waits until next time the clock passes a given local time, then returns. This method is intended for applications, which want to repeat an operation at a given time every day.

Time
The time on any date when the method will return. If this variable has a Date part it is ignored.

WriteLog (Text As String) As Long

Writes a text in the script object's log window, if it is in the SetReportLog() status, which is the default.

Text
A string to be appended to the current tree view item in the log