FILESYSTEM Object

The FILESYSTEM object is an intrinsic object that provides an interface for common file-system operations, such as copying, moving, and deleting files and directories. It is automatically created at startup and is always available.

Developer Notes

1. The FILESYSTEM object cannot be destroyed by the SYSTEM DESTROY method.

FILESYSTEM Properties

The FILESYSTEM object supports the following properties:

Name	Description
CURRENTDIR	Gets or Sets the "Current Directory" for the PS.
DRIVELIST	Returns a list of mapped drives.
FILEOPRESULT	Returns the status of the last file operation method.
SYSTEMDIR	Returns the Windows system directory.
TEMPDIR	Returns the Windows temporary directory for the current user.
WINDOWSDIR	Returns the Windows directory

CURRENTDIR property

Description

Gets or sets the "Current Directory" for the PS process.

Property Value

This property is a string containing a valid operating system path.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get/Set	No	No	No

Remarks

You should exercise some care when setting the current directory using this property as it may cause issues in subsequent data access operations in Basic+, due to the fact that:

1. RevEngine uses the current directory as part of the search path when loading other modules, and
2. The current directory is also used to resolve relative paths.

The best policy is to restore it to its original value as soon as possible if changed.

This property is basically a wrapper around the GetCurrentDirectory and SetCurrentDirectory Windows API functions, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the current directory
CurrentDir = Get_Property( "FILESYSTEM", "CURRENTDIR" )

// Set the current directory
OrigDir = Get_Property( "FILESYSTEM", "CURRENTDIR", "c:\temp" )
```

See also

N/a.

DRIVELIST property

Description

Returns an array containing the letters of the currently available mapped drives.

Property Value

This property is an @Fm-delimited array of drive letters.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

This property is essentially a wrapper around the GetLogicalDrives Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the current list of mapped drive letters
DriveList = Get_Property( "FILESYSTEM", "DRIVELIST" )
```

See also

N/a.

FILEOPRESULT property

Description

Returns an array containing a code and description that describes the result of executing a prior FILESYSTEM method. The following methods set the FILEOPRESULT property when they are executed:

- COPYDIR
- COPYFILES
- DELETEFILES
- GETLONGPATH
- GETSHORTPATH
- MAKEDIR
- MOVEDIR
- MOVEFILES
- REMOVEDIR
- RENAMEDIR
- RENAMEFILE

Property Value

This property is an @Fm-delimited array containing a numeric code and a text description of the result like so:

<1> Code
<2> Description

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

Equated constants for use with the FILEOPRESULT property can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Make a directory and check the error details if it fails
$Insert PS_FileSystem_Equates

If Exec_Method( "FILESYSTEM", "MAKEDIR", "c:\temp\newdir" ) Else

   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>

End
```

See also

FILESYSTEM COPYDIR method, FILESYSTEM COPYFILES method, FILESYSTEM DELETEFILES method, FILESYSTEM GETLONGPATH method, FILESYSTEM GETSHORTPATH method, FILESYSTEM MAKEDIR method, FILESYSTEM MOVEDIR method, FILESYSTEM MOVEFILES method, FILESYSTEM REMOVEDIR method, FILESYSTEM RENAMEDIR method, FILESYSTEM RENAMEFILE method

SYSTEMDIR property

Description
Returns the path for the Windows "system directory". The system directory contains system files such as dynamic-link libraries and drivers.

Property Value
This property is a string containing a valid operating system path.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	N/a	No

Remarks
This property is a simple wrapper around the GetSystemDirectory Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the system directory
SystemDir = Get_Property( "FILESYSTEM", "SYSTEMDIR" )
```

See also
N/a.

TEMPDIR property

Description

Returns the path of the directory designated for temporary files for the current user.

Property Value

This property is a string containing a valid operating system path.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	No	No

Remarks

This property is a simple wrapper around the GetTempPath Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the user's Temp directory
TempDir = Get_Property( "FILESYSTEM", "TEMPDIR" )
```

See also

N/a.

WINDOWSDIR property

Description
Returns the path for the "Windows directory", which specifies the location where the core Windows OS files are installed.

Property Value
This property is a string containing a valid operating system path.

Property Traits

Development	*Runtime*	*Indexed*	*Scaled*	*Synthetic*
N/a	Get	No	N/a	No

Remarks
This property is a simple wrapper around the GetWindowsDirectory Windows API function, so please refer to the documentation on the Microsoft website for further information.

Example

```
// Get the Windows directory
WindowsDir = Get_Property( "FILESYSTEM", "WINDOWSDIR" )
```

See also
N/a.

FILESYSTEM Methods

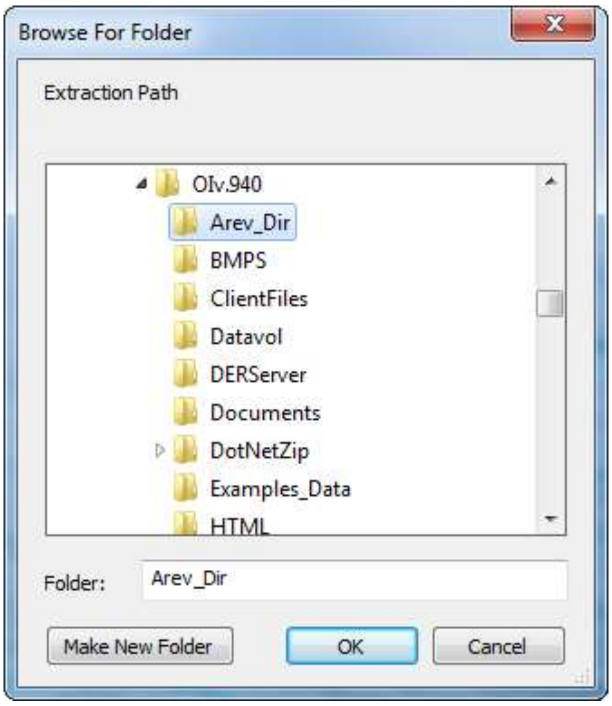
The FILESYSTEM object supports the following methods:

Name	Description
CHOOSEDIR	Executes the Windows Common Browse Folder dialog to select a folder name.
CHOOSEFILE	Executes the Windows Common File dialog to select a file name.
COPYDIR	Copies the contents of a directory to another directory.
COPYFILES	Copies one or more files to another location.
DELETEFILES	Deletes one or more files.
DIREXISTS	Checks to see if a specified directory exists.
FILEEXISTS	Checks to see if a specified file exists.
GETABSOLUTEPATH	Resolves a relative path to an absolute one.
GETLONGPATH	Resolves a short (8.3) path to a long one.
GETRELATIVEPATH	Resolves an absolute path to a relative one.
GETSHORTPATH	Resolves a long path to a short (8.3) form.
GETSPECIALDIR	Returns a specified Windows “special” directory.
MAKEDIR	Creates a new directory at the specified location.
MOVEDIR	Moves a directory and its contents to a new location.
MOVEFILES	Moves one or more files to a new location.
REMOVEDIR	Deletes a specified directory and its contents.
RENAMEDIR	Renames a specified directory to a new name.
RENAMEFILE	Renames a specified file to a new name.

CHOOSEDIR Method

Description

Displays the "Select Folder" Windows Common Dialog Box to allow the user to select a directory or folder name:



Syntax

```
DirName = Exec_Method( "FILESYSTEM",     |
                       "CHOOSEDIR",      |
                       OwnerWindow,      |
                       ChooseDirOptions )
```

Parameters

Name	Required	Description
OwnerWindow	No	ID of the parent form for the dialog. This can be null, in which case the parent form is the desktop.

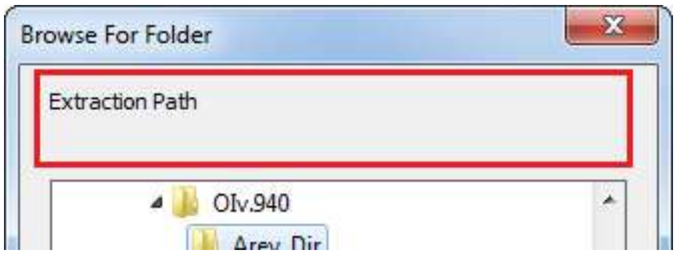
ChooseDirOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the dialog. It has the following structure:
		
		`<1> Title`
		`<2> Initial Directory`
		`<3> Hide new folder button`
		`<4> Show Files`
		
		(See Remarks below for more details)

Returns

A string containing the name and path of the selected folder, or null if the user cancels the dialog.

Remarks

The ChooseDirOptions argument is composed of four fields which control the behavior of the dialog – each field is described fully below:

Field	Name	Description
<1>	Title	Text to display in the title area of the dialog just above the list of folders: 
<2>	Initial Directory	Specifies the initial directory to display when the dialog is created.
<3>	Hide New Folder	If TRUE$ then the "Make New Folder" button is hidden.
<4>	Show Files	If TRUE$ then files are displayed in the dialog as well as folders.

The CHOOSEDIR method is basically a wrapper around the SHBrowserForFolder Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Equated constants for use with the CHOOSEDIR method can be found in the PS_CHOOSEDIR_EQUATES insert record.

Example

```
// Example - get the default folder from some config settings and
// ask the user to choose where they wish to deploy an RDK to.
$Insert PS_FileSystem_Equates

RDKDir = RTI_IDE_Cfg( "GETVALUE", IDE_CFG_RDKINSTALL_DIR$, Drive() )

// IDE104: Please select the folder containing the RDK module you wish to install
ChdArgs = ""
ChdArgs<PS_CHD_TITLE$>          = RTI_Res2Str( RESID$, "IDE104", "", TRUE$ )
ChdArgs<PS_CHD_INITIALDIR$>     = RdkDir
ChdArgs<PS_CHD_HIDENEWFOLDER$> = TRUE$

RdkDir = Exec_Method( "FILESYSTEM", "CHOOSEDIR", @Window, ChdArgs )
If BLen( RdkDir ) Then
   // Export to here...
End
```

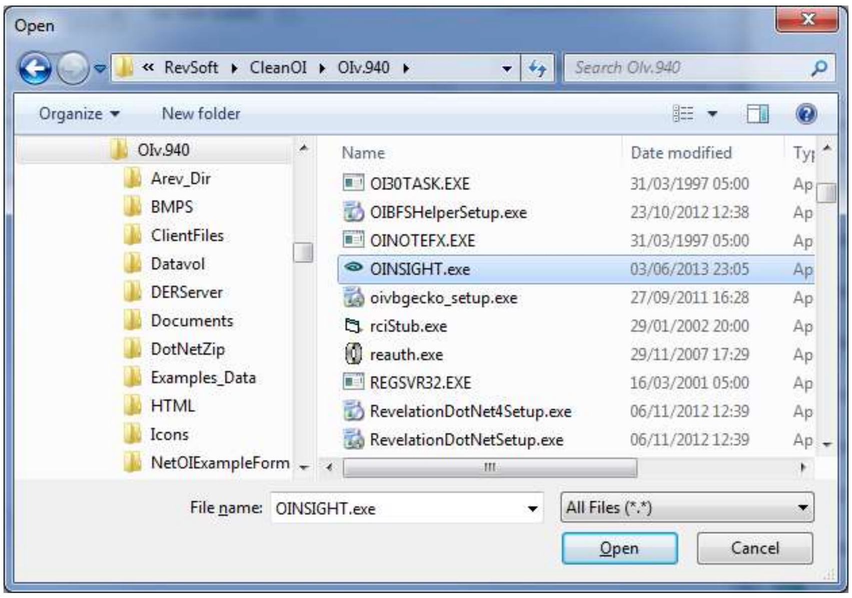
See also

N/a.

CHOOSEFILE Method

Description

Displays the "Open File" or "Save As" Windows Common Dialog Box to allow the user to select a file name:



Name	Date modified	Typ
OI30TASK.EXE	31/03/1997 05:00	Ap
OIBFSHelperSetup.exe	23/10/2012 12:38	Ap
OINOTEFX.EXE	31/03/1997 05:00	Ap
OINSIGHT.exe	03/06/2013 23:05	Ap
oivbgecko_setup.exe	27/09/2011 16:28	Ap
rciStub.exe	29/01/2002 20:00	Ap
reauth.exe	29/11/2007 17:29	Ap
REGSVR32.EXE	16/03/2001 05:00	Ap
RevelationDotNet4Setup.exe	06/11/2012 12:39	Ap
RevelationDotNetSetup.exe	06/11/2012 12:39	Ap

File name: OINSIGHT.exe

All Files (*.*)

Syntax

```
FileName = Exec_Method( "FILESYSTEM",        |
                        "CHOOSEFILE",        |
                        OwnerWindow,         |
                        ChooseFileOptions )
```

Parameters

Name	Required	Description
OwnerWindow	No	ID of the parent form for the dialog. This can be null, in which case the parent form is the desktop.

ChooseFileOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the dialog. It has the following structure:
		
		`<1> Mode`
		`<2> Filter Mask`
		`<3> Filter Index`
		`<4> Default File Name`
		`<5> Flags`
		`<6> Initial Directory`
		`<7> Default Extension`
		`<8> Title`
		
		(See Remarks below for more details)

Returns

If the dialog used in single- select mode the full path and name of the selected file is returned.

If the dialog is used in multi-select mode (i.e. the "Flags" option has the OFN_ALLOWMULTISELECT$ flag set) then an @Fm-delimited dynamic array is returned: The first field in the returned list is the selected directory name, while subsequent items are the selected files, i.e.:

```
<1> Selected Directory
<2> First file
<3> Second file
      :
      :
<n> (n-1)'th file
```

If the user chooses "Cancel" then an empty string is returned.

Remarks

The ChooseFIleOptions argument is composed of eight fields which control the behavior of the dialog – each field is described fully below:

Field	Name	Description
<1>	Mode	Specifies the dialog mode ("Open" or "Save As") bypassing one of the following values: "0" – Open (default mode) "1" – Save As

<2>	Filter String	Contains a list of items (file types and filter masks) that determine what files types the dialog will show.
		
		• Each item in the list is composed of two elements (delimited by a "/" character):
		    o Text to display in the dialog options.
		    o A set of filters that determine what files to display. Each filter is delimited by a ";" character.
		• Each item in the list is also delimited by a /" character
		• The list must be terminated by a "/" character
		
		For example, if you wanted to show bmp files, jpg/jpeg files, and "All files" your filter string would look like this:
		
		`Filter =  "Bitmap Files (*.bmp)/*.bmp/"`
		`Filter := "Jpeg Files (*.jpg,*.jpeg)/*.jpg;*.jpeg/"`
		`Filter := "All Files (*.*)/*.*/"`

		Note that some OS-file based repository types (such as images) have equated constants for filter strings. See the REPOS_IMAGE_EQUATES insert record for an example.
<3>	Filter Index	Contains the index of the initial filter string item to use in the dialog. Defaults to 1.
<4>	Default File Name	Contains the name of the file to pre-select in the dialog when it is displayed.
<5>	Flags	A set of bitmask flags that specify the dialog behavior. These are fully described in the Microsoft documentation for the "flags" member of the OPENFILENAME structure, and equates constants for these flags can be found in the MSWIN_GETOPENFILENAME_EQUATES insert record (See below for an example of using them).
<6>	Initial Directory	Specifies the initial directory to display when the dialog is created.
<7>	Default Extension	Specifies the default extension to append to the file if the user does not specify one. This string can be any length, but only the first three characters are appended. The string should not contain a period (.).
<8>	Title	Caption bar text for the dialog to display.

The CHOOSEFILE method is basically a wrapper around two Windows API functions:

- GetOpenFileName
- GetSaveFileName

It is worth examining the documentation for these functions on the Microsoft website to get a better idea of the capabilities of this method.

Equated constants for use with the CHOOSEFILE method can be found in the PS_CHOOSEFILE_EQUATES and MSWIN_CHOOSEFILE_EQUATES insert records.

Example

```
// Use the CHOOSEFILE method to allow the user to select multiple files...
$Insert PS_ChooseFile_Equates
$Insert Repos_Image_Equates

// Show Bitmaps, JPegs and All Files ...
Filter =  "Bitmap Files(*.bmp)/*.bmp/"
Filter := "JPeg Files (*.jpg,*jpeg,*.jpe)/*.jpg;*jpeg;*.jpe/"
Filter := "All Files (*.*)/*.*/"

// Allow multiple selections, and ensure that any files
// we select exist - we do this by using the OFN_ flags
Flags = 0;
Flags = BitAnd( Flags, OFN_ALLOWMULTISELECT$ )
Flags = BitAnd( Flags, OFN_FILEMUSTEXIST$ )

CFOptions    = ""
CFOptions<1> = CHFILE_MODE_OPEN$
CFOptions<2> = Filter
CFOptions<3> = 1
CFOptions<4> = ""
CFOptions<5> = Flags
CFOptions<6> = Drive() : "\bmps"
CFOptions<7> = ""
CFOptions<8> = "Select Image"

FileList = Exec_Method( "SYSTEM", "CHOOSEFILE", @Window, CFOptions )
If Len( FileList ) Then
   // We used a multi-select dialog so the first item in FileList
   // is the directory, followed by a list of files
   FileDir  = FileList[1,@fm]
   FileList = FileList[Col2()+1,\00\]
End
```

See also

N/a.

COPYDIR Method

Description

Copies a directory and subdirectories to a specified location, optionally showing progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",  |
                           "COPYDIR",     |
                           DirFrom,       |
                           DirTo,         |
                           CopyDirOptions )
```

Parameters

Name	Required	Description
DirFrom	Yes	Fully qualified path of the directory to copy.
DirTo	Yes	Fully qualified path of the location to copy the directory to. Wildcard characters are not supported in this argument.

CopyDirOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the copy operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the copy was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Check the FILEOPRESULT property for error details.

Remarks

The CopyDirOptions argument is composed of five fields which control the behavior of the copy operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.

<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the copy operation.
<4>	Progress dialog title	Text to display in the progress dialog during the copy operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the copy operation. Defaults to FALSE$.

The COPYDIR method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Fully qualified path names should be used with this method. Using it with relative path names is not thread safe.

Equated constants for use with the COPYDIR method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM COPYDIR method to copy an entire folder and it's
// contents to another location, showing any progress information, but not
// allowing the user to override any conflicts.
$Insert PS_FileSystem_Equates
$Insert Logical

DirFrom     = "c:\my_data\aug_2017"
DirTo       = "c:\backup\backup_data"

CopyOptions = ""
CopyOptions<PS_CPD_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
CopyOptions<PS_CPD_ALLOWUI$>        = TRUE$  ; // Show dialogs
CopyOptions<PS_CPD_PARENTWINDOW$>   = @Window
CopyOptions<PS_CPD_PROGRESSTITLE$> = "Copying data to backup folder"
CopyOptions<PS_CPD_ALLOWCONFIRM$>   = FALSE$

If Exec_Method( "FILESYSTEM", "COPYDIR", DirFrom, DirTo, CopyOptions ) Then
   // Copied OK - data is now in:
   //
   //       "c:\backup\backup_data\aug_2017"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>

   // Handle Error  ....
End
```

See also

FILESYSTEM FILEOPRESULT property.

COPYFILES Method

Description

Copies files to a specified location, optionally showing a progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",     |
                           "COPYFILES",      |
                           FilesFrom,        |
                           FilesTo,          |
                           CopyFilesOptions )
```

Parameters

Name	Required	Description
FilesFrom	Yes	@Fm-delimited list of files to copy. Wildcard characters are allowed.
FilesTo	Yes	Fully qualified path of the location to copy the files to, or an @fm-delimited list of destination files. In the latter case the number of file names must match *exactly* the number of file names specified in the "FilesFrom" parameter.

CopyFilesOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the copy operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the copy was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Check the FILEOPRESULT property for error details.

Remarks

The CopyFilesOptions argument is composed of five fields which control the behavior of the copy operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.
<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the copy operation.
<4>	Progress dialog title	Text to display in the progress dialog during the copy operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the copy operation. Defaults to FALSE$.

The COPYFILES method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining at the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Fully qualified path names should be used with this method. Using it with relative path names is not thread safe.

Equated constants for use with the COPYFILES method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM COPYFILES method to copy all files in a folder to
// a specified destination folder using wildcard characters, showing any
// progress information, but not allowing the user to override any
// conflicts.
$Insert PS_FileSystem_Equates
$Insert Logical

FilesFrom     = "c:\my_data\aug_2017\*.*"
FilesTo       = "c:\backup\backup_data\aug_2017"

CopyOptions = ""
CopyOptions<PS_CPF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
CopyOptions<PS_CPF_ALLOWUI$>        = TRUE$  ; // Show dialogs
CopyOptions<PS_CPF_PARENTWINDOW$>  = @Window
CopyOptions<PS_CPF_PROGRESSTITLE$> = "Copying data to backup folder"
CopyOptions<PS_CPF_ALLOWCONFIRM$>  = FALSE$

If Exec_Method( "FILESYSTEM", "COPYFILES", FilesFrom, FilesTo, CopyOptions ) Then
   // Copied OK - data is now in: "c:\backup\backup_data\aug_2017"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End

// Use the FILESYSTEM COPYFILES method to copy specfic files in a folder to
// a specified destination folder showing any progress information, but not
// allowing the user to override any conflicts.
DirFrom       = "c:\my_data\aug_2017\"
DirTo         = "c:\backup\backup_data\aug_2017\"
FileNames     = "book1.xls"
FileNames<-1> = "book2.xls"
FilesFrom     = ""
FilesTo       = ""

XCount = FieldCount( FileNames, @fm )
For X = 1 To XCount
   FilesFrom<x> = DirFrom : FileNames<x>
   FilesTo<x>   = DirTo   : FileNames<x>
Next

CopyOptions  = ""
CopyOptions<PS_CPF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
CopyOptions<PS_CPF_ALLOWUI$>        = TRUE$  ; // Show dialogs
CopyOptions<PS_CPF_PARENTWINDOW$>  = @Window
CopyOptions<PS_CPF_PROGRESSTITLE$> = "Copying data to backup folder"
CopyOptions<PS_CPF_ALLOWCONFIRM$>  = FALSE$

If Exec_Method( "FILESYSTEM", "COPYFILES", FilesFrom, FilesTo, CopyOptions ) Then
   // Copied OK - data is now in: "c:\backup\backup_data\aug_2017"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM FILEOPRESULT property.

DELETEFILES Method

Description

Deletes a set of specified files, optionally showing a progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",     |
                           "DELETEFILES",    |
                           FilesList,        |
                           DelFilesOptions )
```

Parameters

Name	Required	Description
FileList	Yes	@Fm-delimited list of files to delete. Wildcard characters are allowed.

DelFilesOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the delete operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the delete was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Check the FILEOPRESULT property for error details.

Remarks

The DelFilesOptions argument is composed of five fields which control the behavior of the delete operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.
<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the delete operation.
<4>	Progress dialog title	Text to display in the progress dialog during the delete operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the delete operation. Defaults to FALSE$.

The DELETEFILES method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the MSDN website to get a better idea of the capabilities of this method.

You should use fully qualified path names with this method. Using it with relative path names is not thread safe.

Equated constants for use with the DELETEFILES method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM DELETEFILES method to delete all files in a folder
// using wildcard characters, showing any progress information, and
// allowing the user to stop the operation.

$Insert PS_FileSystem_Equates
$Insert Logical

FileList     = "c:\my_data\aug_2017\*.*"

DelOptions = ""
DelOptions<PS_DLF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
DelOptions<PS_DLF_ALLOWUI$>        = TRUE$  ; // Show dialogs
DelOptions<PS_DLF_PARENTWINDOW$>  = @Window
DelOptions<PS_DLF_PROGRESSTITLE$> = "Copying data to backup folder"
DelOptions<PS_DLF_ALLOWCONFIRM$>  = TRUE$

If Exec_Method( "FILESYSTEM", "DELETEFILES", FileList, DelOptions ) Then
   // Deleted OK
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End

// Use the FILESYSTEM DELETEFILES method to delete specific files in a folder
// showing progress information, but not allowing the user to cancel the
// operation.

DirFrom       = "c:\my_data\aug_2017\"

FileNames     = "book1.xls"
FileNames<-1> = "book2.xls"

FileList      = ""

XCount = FieldCount( FileNames, @fm )
For X = 1 To XCount
   FileList<x> = DirFrom : FileNames<x>
Next

DelOptions = ""
DelOptions<PS_CPF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
DelOptions<PS_CPF_ALLOWUI$>        = TRUE$  ; // Show dialogs
DelOptions<PS_CPF_PARENTWINDOW$>  = @Window
DelOptions<PS_CPF_PROGRESSTITLE$> = "Copying data to backup folder"
DelOptions<PS_CPF_ALLOWCONFIRM$>  = FALSE$

If Exec_Method( "FILESYSTEM", "DELETEFILES", FileList, DelOptions ) Then
   // Deleted OK
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM FILEOPRESULT property.

DIREXISTS Method

Description

Determines if a specified directory exists.

Syntax

```
ExistsFlag = Exec_Method( "FILESYSTEM",  |
                          "DIREXISTS",   |
                          DirName )
```

Parameters

Name	Required	Description
DirName	Yes	Specifies the path of the directory to check.

Returns

TRUE$ if the directory exists, or FALSE$ if it doesn't.

Remarks

N/a.

Example

```
// Use the FILESYSTEM DIREXISTS method to check if a directory exists
DirName = "c:\my_data\aug_2017"

If Exec_Method( "FILESYSTEM", "DIREXISTS", DirName ) Then
   // It's there - Process the directory ...
End
```

See also

N/a.

FILEEXISTS Method

Description

Determines if a specified file exists.

Syntax

```
ExistsFlag = Exec_Method( "FILESYSTEM",  |
                          "FILEEXISTS",  |
                          FileName )
```

Parameters

Name	Required	Description
FileName	Yes	Specifies the name and path of the file to check for.

Returns

TRUE$ if the file exists, or FALSE$ if it doesn't.

Remarks

N/a.

Example

```
// Use the FILESYSTEM FILEEXISTS method to check if a file exists
   FileName = "c:\my_data\aug_2017\book3.xls"

If Exec_Method( "FILESYSTEM", "FILEEXISTS", FileName ) Then
   // It's there ...
End
```

See also

N/a.

GETABSOLUTEPATH Method

Description

Resolves a relative path to an absolute one.

Syntax

```
AbsPath = Exec_Method( "FILESYSTEM",       |
                       "GETABSOLUTEPATH", |
                       RelativePath,      |
                       RelativeToPath )
```

Parameters

Name	Required	Description
RelativePath	Yes	Specifies the relative path to resolve.
RelativeToPath	No	Specifies the reference path for the resolution. If this parameter is null then the current directory is used.

Returns

The absolute path if successful, or null if the relative path cannot be resolved.

Remarks

This method does not check if the returned absolute path actually exists.

Example

```
// Use GETABSOLUTEPATH to resolve a relative directory to
// the current directory
//
// (Assume current directory is "c:\revsoft\openinsight")
//
RelPath = "..\my_data"
AbsPath = Exec_Method( "FILESYSTEM", "GETABSOLUTEPATH", RelPath, "" )

// AbsPath should be: "c:\revsoft\my_data"

// Use GETABSOLUTEPATH to resolve a relative directory to
// a specified reference directory
RelPath = "\my_root_dir"
RefPath = "c:\revsoft\openinsight\apps"
AbsPath = Exec_Method( "FILESYSTEM", "GETABSOLUTEPATH", RelPath, RefPath)

// AbsPath should be: "c:\my_root_dir"
```

See also

FILESYSTEM GETRELATIVEPATH method.

GETLONGPATH Method

Description

Converts a legacy DOS short "8.3" style file path to a full one.

Syntax

```
LongPath = Exec_Method( "FILESYSTEM",  |
                        "GETLONGPATH", |
                        ShortPath )
```

Parameters

Name	Required	Description
ShortPath	Yes	Specifies the short path to convert.

Returns

The converted long path if successful, or null if the method fails. The FILEOPRESULT property may be used to obtain more details regarding the failure.

Remarks

The GETLONGPATH method is basically a wrapper around the GetLongPath Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Example

```
// Use GETLONGPATH to convert a short path
ShortPath = "c:\progra~1\my_app"
LongPath  = Exec_Method( "FILESYSTEM", "GETLONGPATH", ShortPath )

If BLen( LongPath ) Then
   // LongPath should be: "C:\Program Files\my_app"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM GETSHORTPATH method, FILESYSTEM FILEOPRESULT property.

GETRELATIVEPATH Method

Description

Resolves an absolute path to a relative one.

Syntax

```
RelPath = Exec_Method( "FILESYSTEM",       |
                       "GETRELATIVEPATH",  |
                       AbsolutePath,       |
                       RelativeToPath )
```

Parameters

Name	Required	Description
AbsolutePath	Yes	Specifies the absolute path to resolve.
RelativeToPath	No	Specifies the reference path for the resolution. If this parameter is null then the current directory is used.

Returns

The relative path if successful, or null if the absolute path cannot be resolved.

Remarks

This method does not check if the returned relative path actually exists.

Example

```
// Use GETRELATIVEPATH to resolve a an absolute directory to
// the current directory
//
// (Assume current directory is "c:\revsoft\openinsight")
//
AbsPath = "c:\revsoft\my_data"
RelPath = Exec_Method( "FILESYSTEM", "GETRELATIVEPATH", AbsPath, "" )

// RelPath should be: "..\my_data"

// Use GETRELATIVEPATH to resolve an absolute directory to
// a specified reference directory
AbsPath = "c:\my_root_dir"
RefPath = "c:\revsoft\openinsight\apps"
RelPath = Exec_Method( "FILESYSTEM", "GETRELATIVEPATH", AbsPath, RefPath)

// RelPath should be: "\my_root_dir"
```

See also

FILESYSTEM GETABSOLUTEPATH method.

GETSHORTPATH Method

Description

Converts a "normal" long path to a legacy short DOS-style short "8.3" path.

Syntax

```
ShortPath = Exec_Method( "FILESYSTEM",    |
                         "GETSHORTPATH",  |
                         LongPath )
```

Parameters

Name	Required	Description
LongPath	Yes	Specifies the long path to convert.

Returns

The converted short path if successful, or null if the method fails. The FILEOPRESULT property may be used to obtain more details regarding the failure.

Remarks

The GETSHORTPATH method is basically a wrapper around the GetShortPath Windows API function, so it is worth examining at the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Example

```
// Use GETSHORTPATH to convert a long path
LongPath  = "C:\Program Files\my_app"
ShortPath = Exec_Method( "FILESYSTEM", "GETSHORTPATH", LongPath )

If BLen( ShortPath ) Then
   // ShortPath should be: "C:\ Progra~1\my_app"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM GETLONGPATH method, FILESYSTEM FILEOPRESULT property.

GETSPECIALDIR Method

Description

Returns the path for a "special" Windows directory, such as the directory to use for temporary files, or the user's "My Documents" directory and so on.

Syntax

```
SpecialDirPath = Exec_Method( "FILESYSTEM",    |
                              "GETSPECIALDIR", |
                              DirID )
```

Parameters

Name	Required	Description
DirID	Yes	Specifies the directory path to return. Can be one of the following values: • "ADMINTOOLS" • "ALTSTARTUP" • "APPDATA" • "COMMON_ADMINTOOLS" • "COMMON_ALTSTARTUP" • "COMMON_APPDATA" • "COMMON_DESKTOPDIRECTORY" • "COMMON_DOCUMENTS" • "COMMON_FAVORITES" • "COMMON_PROGRAMS" • "COMMON_STARTMENU" • "COMMON_STARTUP" • "COMMON_TEMPLATES" • "COOKIES" • "DESKTOPDIRECTORY" • "FAVORITES" • "FONTS" • "HISTORY" • "INTERNET_CACHE" • "LOCAL_APPDATA" • "MYMUSIC" • "MYPICTURES" • "NETHOOD" • "PERSONAL" • "PRINTHOOD" • "PROFILE" • "PROGRAM_FILES" • "PROGRAM_FILES_COMMON" • "PROGRAMS" • "RECENT" • "SENDTO" • "STARTMENU" • "STARTUP" • "SYSTEM" • "TEMPLATES" • "WINDOWS"

Returns

The specified path.

Remarks

The GETSPECIALDIR method is a simple wrapper around the SHGetSpecialFolderPath Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Equated constants for use with the GETSPECIALDIR method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
   // Return the current user's "My Documents" folder
   MyDocsDir = Exec_Method( "FILESYSTEM", "GETSPECIALDIR", "PERSONAL" )
```

See also

FILESYSTEM TEMPDIR property, FILESYSTEM SYSTEMDIR property, FILESYSTEM WINDOWSDIR property

MAKEDIR Method

Description
Creates a new directory at the specified location.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",   |
                           "MAKEDIR",      |
                           DirName,        |
                           RecurseFlag )
```

Parameters

Name	Required	Description
DirName	Yes	Specifies the path of the directory to create.
RecurseFlag	No	If TRUE$ then recurse to create any subdirectories as needed. Defaults to FALSE$.

Returns
TRUE$ if the directory was created successfully (or already exists), or FALSE$ an error occurred. The FILEOPRESULT property may be used to obtain more details regarding the failure.

Remarks
N/a.

Example

```
// Use the FILESYSTEM MAKEDIR method to create a directory
DirName = "c:\my_data\aug_2017"

If Exec_Method( "FILESYSTEM", "MAKEDIR", DirName ) Then
   // It's there - Process the directory ...
End
```

See also
N/a.

MOVEDIR Method

Description

Moves a directory and subdirectories to a specified location, optionally showing progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",   |
                           "MOVEDIR",      |
                           DirFrom,        |
                           DirTo,          |
                           MoveDirOptions )
```

Parameters

Name	Required	Description
DirFrom	Yes	Fully qualified path of the directory to move.
DirTo	Yes	Fully qualified path of the location to move the directory to. Wildcard characters are not supported in this argument.

MoveDirOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the move operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the move was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Use the FILEOPRESULT property for more details regarding any failure.

Remarks

The MoveDirOptions argument is composed of five fields which control the behavior of the move operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.
<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the move operation.
<4>	Progress dialog title	Text to display in the progress dialog during the move operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the move operation. Defaults to FALSE$.

The MOVEDIR method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Fully qualified path names should be used with this method. Using it with relative path names is not thread safe.

Equated constants for use with the MOVEDIR method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM MOVEDIR method to move an entire folder and it's
// contents to another location, showing any progress information, but not
// allowing the user to override any conflicts.

$Insert PS_FileSystem_Equates
$Insert Logical

DirFrom     = "c:\my_data\aug_2017"
DirTo       = "c:\backup\backup_data"

MoveOptions = ""
MoveOptions<PS_MVD_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
MoveOptions<PS_MVD_ALLOWUI$>        = TRUE$  ; // Show dialogs
MoveOptions<PS_MVD_PARENTWINDOW$>   = @Window
MoveOptions<PS_MVD_PROGRESSTITLE$> = "Moving data to backup folder"
MoveOptions<PS_MVD_ALLOWCONFIRM$>   = FALSE$

If Exec_Method( "FILESYSTEM", "MOVEDIR", DirFrom, DirTo, MoveOptions ) Then
   // Moved OK - data is now in:
   //
   //       "c:\backup\backup_data\aug_2017"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )

   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>

   // Handle Error  ....
End
```

See also

FILESYSTEM FILEOPRESULT property.

MOVEFILES Method

Description

Moves files to a specified location, optionally showing progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",     |
                           "MOVEFILES",      |
                           FilesFrom,        |
                           FilesTo,          |
                           MoveFilesOptions )
```

Parameters

Name	Required	Description
FilesFrom	Yes	@Fm-delimited list of files to move. Wildcard characters are allowed.
FilesTo	Yes	Fully qualified path of the location to move the files to, or an @Fm-delimited list of destination files. In the latter case the number of file names must match *exactly* the number of file names specified in the "FilesFrom" parameter.

MoveFilesOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the move operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the move was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Use the FILEOPRESULT property for more details regarding any failure.

Remarks

The MoveFilesOptions argument is composed of five fields which control the behavior of the move operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.
<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the move operation.
<4>	Progress dialog title	Text to display in the progress dialog during the move operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the move operation. Defaults to FALSE$.

The MOVEFILES method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Fully qualified path names should be used with this method. Using it with relative path names is not thread safe.

Equated constants for use with the MOVEFILES method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM MOVEFILES method to move all files in a folder to
// a specified destination folder using wildcard characters, showing any
// progress information, but not allowing the user to override any
// conflicts.
$Insert PS_FileSystem_Equates
$Insert Logical

FilesFrom     = "c:\my_data\aug_2017\*.*"
FilesTo       = "c:\backup\backup_data\aug_2017"

MoveOptions = ""
MoveOptions<PS_MVF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
MoveOptions<PS_MVF_ALLOWUI$>        = TRUE$  ; // Show dialogs
MoveOptions<PS_MVF_PARENTWINDOW$>  = @Window
MoveOptions<PS_MVF_PROGRESSTITLE$> = "Moving data to backup folder"
MoveOptions<PS_MVF_ALLOWCONFIRM$>  = FALSE$

If Exec_Method( "FILESYSTEM", "MOVEFILES", FilesFrom, FilesTo, MoveOptions ) Then
   // Moved OK - data is now in: "c:\backup\backup_data\aug_2017"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End

// Use the FILESYSTEM MOVEFILES method to move specific files in a folder to
// a specified destination folder showing any progress information, but not
// allowing the user to override any conflicts.
DirFrom       = "c:\my_data\aug_2017\"
DirTo         = "c:\backup\backup_data\aug_2017\"
FileNames     = "book1.xls"
FileNames<-1> = "book2.xls"
FilesFrom     = ""
FilesTo       = ""

XCount = FieldCount( FileNames, @fm )
For X = 1 To XCount
   FilesFrom<x> = DirFrom : FileNames<x>
   FilesTo<x>   = DirTo   : FileNames<x>
Next

MoveOptions  = ""
MoveOptions<PS_MVF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
MoveOptions<PS_MVF_ALLOWUI$>        = TRUE$  ; // Show dialogs
MoveOptions<PS_MVF_PARENTWINDOW$>  = @Window
MoveOptions<PS_MVF_PROGRESSTITLE$> = "Moving data to backup folder"
MoveOptions<PS_MVF_ALLOWCONFIRM$>  = FALSE$

If Exec_Method( "FILESYSTEM", "MOVEFILES", FilesFrom, FilesTo, MoveOptions ) Then
   // Moved OK - data is now in: "c:\backup\backup_data\aug_2017"
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM FILEOPRESULT property.

REMOVEDIR Method

Description

Removes a specified directory, optionally showing progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",     |
                           "REMOVEDIR",      |
                           DirName,          |
                           RemoveDirOptions )
```

Parameters

Name	Required	Description
DirName	Yes	Specifies the path of the directory to remove.

RemoveFilesOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the remove operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the directory was successfully removed, or FALSE$ if an error occurred or the operation was cancelled. Use the FILEOPRESULT property for more details regarding any failure.

Remarks

The REMOVEDIR method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Fully qualified path names should be used with this method. Using it with relative path names is not thread safe.

Equated constants for use with the REMOVEDIR method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM REMOVEDIR method to remove an entire folder and it's
// contents  showing any progress information, but not allowing the user to
// override any conflicts.

$Insert PS_FileSystem_Equates
$Insert Logical

DirName = "c:\my_data\aug_2017"

RemoveOptions = ""
RemoveOptions<PS_RMD_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
RemoveOptions<PS_RMD_ALLOWUI$>        = TRUE$  ; // Show dialogs
RemoveOptions<PS_RMD_PARENTWINDOW$>   = @Window
RemoveOptions<PS_RMD_PROGRESSTITLE$> = "Removing data folder"
RemoveOptions<PS_RMD_ALLOWCONFIRM$>   = FALSE$

If Exec_Method( "FILESYSTEM", "REMOVEDIR", DirName, RemoveOptions ) Then
   // Removed OK
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )

   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>

   // Handle Error  ....
End
```

See also

N/a.

RENAMEDIR Method

Description

Renames a directory, optionally showing progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",     |
                           "RENAMEDIR",      |
                           DirOldName,       |
                           DirNewName,       |
                           RenameDirOptions )
```

Parameters

Name	Required	Description
DirFrom	Yes	Fully qualified path of the directory to rename.
DirTo	Yes	Fully qualified path of the new directory name. Wildcard characters are not supported in this argument.

RenameDirOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the rename operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the rename was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Use the FILEOPRESULT property obtain more details regarding a failure.

Remarks

The RenameDirOptions argument is composed of five fields which control the behavior of the rename operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.

<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the rename operation.
<4>	Progress dialog title	Text to display in the progress dialog during the rename operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the rename operation. Defaults to FALSE$.

The RENAMEDIR method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Fully qualified path names should be used with this method. Using it with relative path names is not thread safe.

Equated constants for use with the RENAMEDIR method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM RENAMEDIR method to rename an entire folder, showing
// any progress information, but not allowing the user to override any
// conflicts.

$Insert PS_FileSystem_Equates
$Insert Logical

DirFrom     = "c:\my_data\aug_2017"
DirTo       = "c:\my_data\aug_2017.bak"

RenameOptions  = ""
RenameOptions<PS_RND_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
RenameOptions<PS_RND_ALLOWUI$>        = TRUE$  ; // Show dialogs
RenameOptions<PS_RND_PARENTWINDOW$>   = @Window
RenameOptions<PS_RND_PROGRESSTITLE$> = "Renaming data to backup folder"
RenameOptions<PS_RND_ALLOWCONFIRM$>   = FALSE$

If Exec_Method( "FILESYSTEM", "RENAMEDIR", DirFrom, DirTo, RenameOptions ) Then
   // Renamed OK
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )
   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM FILEOPRESULT property.

RENAMEFILE Method

Description

Renames a specified file, optionally showing progress information and allowing an Undo operation.

Syntax

```
SuccessFlag = Exec_Method( "FILESYSTEM",      |
                           "RENAMEFILE",      |
                           FileFrom,          |
                           FileTo,            |
                           RenameFileOptions )
```

Parameters

Name	Required	Description
FileFrom	Yes	Fully qualified path containing the file to rename (wildcard characters are not allowed).
FileTo	Yes	Fully qualified path containing the file to rename.

RenameFileOptions	No	Contains an @Fm-delimited dynamic array of options that control the behavior of the rename operation:
		
		`<1> Allow undo`
		`<2> Allow UI`
		`<3> Parent window ID`
		`<4> Progress dialog title`
		`<5> Allow confirmations`
		
		(See Remarks below for more details)

Returns

TRUE$ if the rename was performed successfully, or FALSE$ if an error occurred or the operation was cancelled. Use the FILEOPRESULT property to obtain more details regarding a failure.

Remarks

The RenameFileOptions argument is composed of five fields which control the behavior of the move operation – each field is described fully below:

Field	Name	Description
<1>	Allow undo	If TRUE$ then save Undo information. Defaults to FALSE$.
<2>	Allow UI	If TRUE$ then allow any appropriate dialogs to be displayed such as progress and error information. Defaults to FALSE$.
<3>	Parent window ID	Specifies the name of a WINDOW object to use as the parent for any dialogs displayed during the rename operation.
<4>	Progress dialog title	Text to display in the progress dialog during the rename operation.
<5>	Allow confirmations	If TRUE$ then allow the user to respond to any confirmation dialogs presented during the rename operation. Defaults to FALSE$.

You cannot rename multiple files with a single call to this method. Use the MOVEFILES method instead.

The RENAMEFILE method is basically a wrapper around the SHFileOperation Windows API function, so it is worth examining the documentation for this on the Microsoft website to get a better idea of the capabilities of this method.

Use fully qualified path names with this method. Using it with relative path names is not thread safe.

Equated constants for use with the RENAMEFILE method can be found in the PS_FILESYSTEM_EQUATES insert record.

Example

```
// Use the FILESYSTEM RENAMEFILE method rename a file, showing any
// progress information, and allowing the user to override any
// conflicts.

$Insert PS_FileSystem_Equates
$Insert Logical

FileFrom = "c:\my_data\aug_2017\book1.xls"
FileTo   = "c:\my_data\aug_2017\book1_backup.xls"

RenameOptions = ""
RenameOptions<PS_RNF_ALLOWUNDO$>      = TRUE$  ; // Save Undo info
RenameOptions<PS_RNF_ALLOWUI$>        = TRUE$  ; // Show dialogs
RenameOptions<PS_RNF_PARENTWINDOW$>   = @Window
RenameOptions<PS_RNF_PROGRESSTITLE$> = "Renaming book1.xls"
RenameOptions<PS_RNF_ALLOWCONFIRM$>   = FALSE$

If Exec_Method( "FILESYSTEM", "RENAMEFILE", FileFrom, FileTo, RenameOptions ) Then
   // Renamed OK
End Else
   ErrorInfo = Get_Property( "FILESYSTEM", "FILEOPRESULT" )

   ErrorCode = ErrorInfo<PS_FOR_ERRORCODE$>
   ErrorText = ErrorInfo<PS_FOR_ERRORTEXT$>
End
```

See also

FILESYSTEM MOVEFILES method, FILESYSTEM FILEOPRESULT property.