Reference Manual

version 2.0

macscript.com

Introduction

What is the Macscript.com Library?

The Macscript.com Library is a collection of 150 AppleScript functions. It allows AppleScript writers to quickly and easily write powerful scripts with only a few lines of code. Each function has been rigorously tested and is optimized for speed.

How do I use the Macscript.com Library?

The Macscript.com Library package contains files for development in a variety of AppleScript editors, including Apple's *Script Editor*, DTI's *FaceSpan*, and Late Night Software's *Script Debugger*. Instructions for using the Library with each of these packages can be found in the readme files within the respective folders included in this package.

Can I use the Library with Mac OS X?

Many of the changes in version 2.0 of the Macscript.com Library have been made to support Mac OS X. Most of the functions work identically under both the classic Mac OS and Mac OS X version 10.1. The exception is the four *ticks* functions (*CheckTicks()*, *ResetTicks()*, *StartTicks()*, and *StopTicks()*) – as these functions require the *Jon's Commands* Scripting Addition by Jon Pugh, which is not yet available for Mac OS X.

We are committed to supporting Mac OS X and we may release updates to the Library as Mac OS X continues to evolve.

For more information

For more information about using the Macscript.com Library, please refer to the example scripts included in this package. If you require more specific information, you can contact support@macscript.com or visit http://www.macscript.com.

Developer Notes

The following points may be of interest to those developing with the Macscript.com Library.

Use of variables

All variables local to each function are explicitly declared as local. This eliminates the possibility of the functions modifying any pre-defined global variables in your project.

The Macscript.com Library also declares a script object called "_MacscriptLib" which contains properties that certain Library functions use.

Reliance on Scripting Additions (OSAXen)

Very few of the functions in the Macscript.com Library rely on any Scripting Additions. Wherever practical, functions are written natively in AppleScript. This approach simplifies things by reducing the need for users to have (and pay for) third-party Scripting Additions, and has the added benefit of dramatically improving performance. Some of the Library's math functions, for example, run 100 times faster than equivalent commands from Scripting Additions.

The exception to this rule is the use of the *Jon's Commands* Scripting Addition by Jon Pugh. Jon's Commands is required to use the *StartTicks()*, *ResetTicks()* and *StopTicks()* functions. Calling any of these functions without Jon's Commands installed will produce an error type -1708 and a message indicating that Jon's Commands is required. Jon's Commands is not needed for any of the other functions in the Macscript.com Library.

Note: Jon's Commands is also not required if the Macscript.com Library is used in a FaceSpan project since FaceSpan has a similar "ticks" function to that of Jon's Commands built in.

Coercions

Each of the functions tries to ensure that the values passed to them are the right class. If a supplied value is not the right class, the function may try to coerce it to the appropriate class and continue rather than just returning an error.

Error handling

The library functions generally do not or handle or trap errors, except when a particular value or condition (such as AppleScript's text item delimiters) should be restored first. For example, if you try to write to a file that doesn't exist by calling *WriteToFile()*, your code will receive a "File not found." error (-43). Passing errors such as these along helps to ensure that subsequent code performs as expected.

Manipulating AppleScript's text item delimiters

Whenever AppleScript's text item delimiters are manipulated within a function, the previous delimiters are stored in a local variable. Any errors are subsequently trapped, and the delimiters returned to their previously saved state before the error is continued. Thus:

```
on DoSomething()
    local theOldDelims

    set theOldDelims to AppleScript's text item delimiters
    set AppleScript's text item delimiters to tab
    try
        --statements

        set AppleScript's text item delimiters to theOldDelims
    on error e number n from f to t partial result p
        set AppleScript's text item delimiters to theOldDelims
        error e number n from f to t partial result p
    end try
end DoSomething
```

Note: Functions that manipulate AppleScript's text item delimiters internally do not affect any text item delimiters previously stored by the Library's *SetDelims()* function.

Intuition

Some Macscript.com Library functions which take parameters will allow the use of various null descriptors (such as **{}**, **""**, and **0**) to indicate that the function should use its default for that parameter. For example, given the following function:

```
CreateFile(theFileSpec, theFileTypeCode, theCreatorCode)
```

supplying a null file type code or creator type code indicates to CreateFile() that it should use its defaults ("TEXT" and "ttxt" respectively); thus:

```
CreateFile(path to desktop, {}, {})
```

will create a new file on the desktop with the file type of "TEXT" and creator type of "ttxt" (a SimpleText file). This functionality is described in the definition of each participating function later in this document.

Plurality

Many of the functions can act on either a single item or a list of items. Passing a singular item to a function will generally return a singular result; passing a plural item (a list of items) to a function will generally return a plural result (a list of result items). For example:

`RoundNumber(2.49, 1)`	returns: 2.5 (singular result)
`RoundNumber({2.49, 3.95}, 1)`	returns: {2.5, 4.0} (plural result)

This functionality is described in the definition of each participating function later in this document.

Use of the Finder

For performance reasons, the Finder is used as little as possible. Only some of the Library's file and process functions use the Finder.

User interaction

Functions which allow user interaction over a given process (e.g. *ChooseFileOverProcess()*) simply tell the target process to perform the interaction. Your code must still manage bringing that process to the foreground (if desired) and handle any timeouts.

Note: Mac OS X (versions 10.0.0–10.0.4) does not allow an AppleScript to tell a particular process to perform this type of user interaction. This has been changed in Mac OS X version 10.1 to work as in classic Mac OS.

Function Definitions

The following pages contain descriptions of each of the functions contained in the Macscript.com Library.

AppendFileName( ) *File functions*

Appends the supplied suffix to the given base file name (truncating the base if required).

Usage

AppendFileName(theBaseName, theSuffix)

Parameters

theBaseName	*string* the base name for the file
theSuffix	*string* the suffix to be appended to the base name; a period character (.) may be inserted between the base name and the suffix (if not there already); if the suffix begins with a space character, the space will be used as the separator

Result

string	the appended file name (up to 31 characters)

Scripting additions required

None

CreateFile( ) *File functions*

Creates a new file with the supplied file type and creator type.

Usage

CreateFile(theFileSpec, theFileTypeCode, theCreatorCode)

Parameters

theFileSpec	*file path, file spec, or alias* location for the file to be created
theFileTypeCode	*string* the four-character file type code for the file to be created (defaults to "TEXT" if supplied as an empty list or string)
theCreatorCode	*string* the four-character creator type code for the file to be created (defaults to "ttxt" (SimpleText) if supplied as an empty list or string)

Result

alias	reference to the newly created file

Scripting additions required

Standard (Read/Write Commands)

CreateFolder( ) *File functions*

Creates the specified folder (and any required parent folders).

Usage

```
CreateFolder(theFolderPath)
```

Parameters

theFolderPath	*file path, file spec, or alias* location for the folder to be created

Result

alias	reference to the newly created folder

Scripting additions required

None

DivideFileName( ) *File functions*

Divides the supplied file name into its base name and its suffix (if any).

Usage

```
DivideFileName(theName)
```

Parameters

theName	*string* the file name to parse

Result

record	record of {base: "*base name*", suffix: "*suffix*"}

Scripting additions required

None

DividePath( ) *File functions*

Divides the supplied file path (or list of file paths) into its name and parent folder path.

Usage

DividePath(thePathList)

Parameters

thePathList	*file path, file spec, or alias (or list thereof)* the file or folder paths to divide

Result

record	record of {name: "*file name*", folder: "*parent folder*"} (or list thereof)

Scripting additions required

None

DoesFileExist( ) *File functions*

Returns whether or not the given file item exists.

Usage

DoesFileExist(theFile)

Parameters

theFile	*file path, file spec, or alias* the file item (file, folder, package, disk) to test for

Result

boolean	*true* if the given file item exists; *false* if it does not

Scripting additions required

None

FilterFiles( ) *File functions*

Filters out all files in the given folder except those of the indicated file and/or creator types.

Usage

FilterFiles(theFolder, showInvisibles, theAllowableTypeList, theAllowableCreatorList)

Parameters

theFolder	*file path, file spec, or alias* the folder to get file list from
showInvisibles	*boolean* whether to include invisible files in the list or not
theAllowableTypeList	*string or list of string* the four-character file type (or list thereof) to be allowed (this parameter is ignored if supplied as an empty list or string)
theAllowableCreatorList	*string or list of string* the four-character creator type (or list thereof) to be allowed (this parameter is ignored if supplied as an empty list or string)

Result

list of alias	list of filtered files

Scripting additions required

Standard (File Commands)

FilterFileList( ) *File functions*

Filters out all files in the given list of files except those of the indicated file and/or creator types.

Usage

FilterFileList(theFileList, theAllowableTypeList, theAllowableCreatorList)

Parameters

theFileList	*list of file path, file spec, or alias* the list of files to filter
theAllowableTypeList	*string or list of string* the four-character file type (or list thereof) to be allowed (this parameter is ignored if supplied as an empty list or string)
theAllowableCreatorList	*string or list of string* the four-character creator type (or list thereof) to be allowed (this parameter is ignored if supplied as an empty list or string)

Result

list of alias	list of filtered files

Scripting additions required

Standard (File Commands)

FinderCopyFile( ) *File functions*

Copies the given file(s) using the Finder.

Usage

FinderCopyFile(theFileList, theDestinationFolder, doReplace)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to copy
theDestinationFolder	*file path, file spec, or alias* the destination folder to copy the file(s) to
doReplace	*boolean* whether to replace existing file(s) or not

Result

alias	reference to the copied file or files

Scripting additions required

None

FinderCreateFile( ) *File functions*

Creates a new file with the supplied file type and creator type using the Finder.
In classic Mac OS, the Finder creates files with both a data fork and a resource fork.

Usage

FinderCreateFile(theFileSpec, theFileTypeCode, theCreatorCode)

Parameters

theFileSpec	*file path, file spec, or alias* the location for the file to be created
theFileTypeCode	*string* the four-character file type code for the file to be created (defaults to "TEXT" if supplied as an empty list or string)
theCreatorCode	*string* the four-character creator type code for the file to be created (defaults to "ttxt" (SimpleText) if supplied as an empty list or string)

Result

alias	reference to the newly created file

Scripting additions required

None

FinderDeleteFile( ) *File functions*

Deletes the given file(s) using the Finder (by moving them to the trash and trying to empty it).

Usage

FinderDeleteFile(theFileList)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to delete

Result

nothing

Scripting additions required

None

FinderMoveFile( ) *File functions*

Moves the given file(s) using the Finder.

Usage

FinderMoveFile(theFileList, theDestinationFolder, doReplace)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to move
theDestinationFolder	*file path, file spec, or alias* the destination folder to move the file(s) to
doReplace	*boolean* whether to replace existing files or not

Result

alias	reference to the moved file or files

Scripting additions required

None

FinderRenameFile( ) *File functions*

Renames the given file using the Finder.

Usage

FinderRenameFile(theFile, theNewName)

Parameters

theFile	*file path, file spec, or alias* the file to rename
theNewName	*string* the new name for the file

Result

alias	reference to the renamed file

Scripting additions required

None

FlushFile( ) *File functions*

Empties the data fork of the given file.

Usage

FlushFile(theFile)

Parameters

theFile	*file path, file spec, or alias* the file to empty

Result

nothing

Scripting additions required

None

GetCreatorType( ) *File functions*

Gets the creator type of the given file (or list of files).

Usage

GetCreatorType(theFileList)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to get the creator type for

Result

string	creator type for each of the given files

Scripting additions required

Standard (File Commands)

GetEOF( ) *File functions*

Returns the offset to the end of file marker in the specified file.

Usage

GetEOF(theFile)

Parameters

theFile	*file path, file spec, or alias* the file to get the end of file marker offset in

Result

integer	offset to the end of file marker; returns -1 on an error

Scripting additions required

Standard (Read/Write Commands)

GetFileAndCreatorType( ) *File functions*

Gets the file and creator type of the given file (or list of files).

Usage

GetFileAndCreatorType(theFileList)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to get the file and creator types for

Result

list	list of {file type, creator type} for each of the given files

Scripting additions required

Standard (File Commands)

GetFileID( ) *File functions*

Gets the file reference number for the specified file.

Usage

GetFileID(theFile)

Parameters

theFile	*file path, file spec, or alias* the file to get the ID of

Result

integer	reference number of the file

Scripting additions required

Standard (Read/Write Commands)

GetFileList( ) *File functions*

Returns the file path to each item within the given volume or folder. See also *ListFolder( )*.

Usage

GetFileList(theFolder, showInvisibles)

Parameters

theFolder	*file path, file spec, or alias* the volume or folder to get file list from
showInvisibles	*boolean* whether to include invisible files in the list or not

Result

list of alias	list of file items

Scripting additions required

Standard (File Commands)

GetFileName( ) *File functions*

Gets the file name(s) of the specified file(s).

Usage

GetFileName(theFileList)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to get name(s) of

Result

string	name of each of the supplied files

Scripting additions required

None

GetFiles( ) *File functions*

Returns the file path to each file (as opposed to each folder) within the given volume or folder. See also *GetFolders( )*.

Usage

GetFiles(theFolder, showInvisibles)

Parameters

theFolder	*file path, file spec, or alias* the volume or folder to get file list from
showInvisibles	*boolean* whether to include invisible files in the list or not

Result

list of alias	list of aliases to the files

Scripting additions required

Standard (File Commands)

GetFilesFromList( ) *File functions*

Returns the file path to each file (as opposed to each folder) within the given list of files. See also *GetFoldersFromList( )*.

Usage

GetFilesFromList(theFileList)

Parameters

theFileList	*list of file path, file spec, or alias* list of files to filter

Result

list of alias	list of aliases to the files

Scripting additions required

Standard (File Commands)

GetFileType( ) *File functions*

Gets the file type of the given file (or list of files).

Usage

```
GetFileType(theFileList)
```

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to get the file type for

Result

string	file type for each of the given files

Scripting additions required

Standard (File Commands)

GetFolders( ) *File functions*

Returns the file path to each folder (as opposed to each file) within the given volume or folder. See also *GetFiles( )*.

Usage

```
GetFolders(theFolder, showInvisibles)
```

Parameters

theFolder	*file path, file spec, or alias* the volume or folder to get folder list from
showInvisibles	*boolean* whether to include invisible folders in the list or not

Result

list of alias	list of aliases to the folders

Scripting additions required

Standard (File Commands)

GetFoldersFromList( ) *File functions*

Returns the file path to each folder (as opposed to each file) within the given list of files.
See also *GetFilesFromList( )*.

Usage

GetFoldersFromList(theFileList)

Parameters

theFileList	*list of file path, file spec, or alias* list of files to filter

Result

list of alias	list of aliases to the folders

Scripting additions required

Standard (File Commands)

GetParentFolder( ) *File functions*

Gets the path to the folder containing each of the given file(s).

Usage

GetParentFolder(theFileList)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to get the parent folder of

Result

string	file path for the parent folder of the file(s)

Scripting additions required

None

IsFileBusy( ) *File functions*

Returns whether the data fork of the given file is busy (unable to be opened for write access).

Usage

IsFileBusy(theFile)

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to test

Result

boolean	*true* if the file is busy; *false* if it is not

Scripting additions required

Standard (Read/Write Commands)

IsFile( ) *File functions*

Returns whether the given file item is a file (as opposed to a folder). See also *IsFolder()* and *IsPackage()*.

Usage

IsFile(theFile)

Parameters

theFile	*file path, file spec, or alias* the file item to test

Result

boolean	*true* if the item is a file; *false* if it is not

Scripting additions required

None

IsFolder( ) *File functions*

Returns whether the given file item is a folder (as opposed to a file). See also *IsFile( )* and *IsPackage( )*.

Usage

IsFolder(theFile)

Parameters

theFile	*file path, file spec, or alias* the file item to test

Result

boolean	*true* if the item is a folder; *false* if it is not

Scripting additions required

None

IsPackage( ) *File functions*

Returns whether the given file item is a Mac OS X package. See also *IsFile( )* and *IsFolder( )*.

Usage

IsPackage(theFile)

Parameters

theFile	*file path, file spec, or alias* the file item to test

Result

boolean	*true* if the item is a package; *false* if it is not

Scripting additions required

Standard (Read/Write Commands)

ListFolder( ) *File functions*

Returns the file path to each item within the given volume or folder. See also *GetFileList( )*.

Usage

ListFolder(theFolder, showInvisibles)

Parameters

theFolder	*file path, file spec, or alias* the volume or folder to get file list from
showInvisibles	*boolean* whether to include invisible files in the list or not

Result

list of alias	list of filtered files

Scripting additions required

Standard (File Commands)

ReadFile( ) *File functions*

Reads the data fork of the given file in its entirety.

Usage

ReadFile(theFile)

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to read

Result

string	the contents of the file

Scripting additions required

Standard (Read/Write Commands)

ReadFileAsClass( ) *File functions*

Reads the data fork of the given file in its entirety, coercing it to the specified class.

Usage

ReadFileAsClass(theFile, theClass)

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to read
theClass	*constant* the class to coerce to

Result

anything	the contents of the file

Scripting additions required

Standard (Read/Write Commands)

ReadFileAsList( ) *File functions*

Reads the data fork of the given file in its entirety, as a list of text items.

Usage

ReadFileAsList(theFile, theDelimiters)

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to read
theDelimiters	*string, or list of up to two strings* the string(s) to delimit the data

Result

list	the text items of the file

Scripting additions required

Standard (Read/Write Commands)

ReadFileFor( ) *File functions*

Reads the data fork of the given file for the specified number of bytes.

Usage

```
ReadFileFor(theFile, theStartByte, theNumberOfBytes)
```

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to read
theStartByte	*integer* the byte to start reading from (inclusive)
theNumberOfBytes	*integer* the number of bytes to read

Result

string	the contents of the file

Scripting additions required

Standard (Read/Write Commands)

ReadFileRange( ) *File functions*

Reads the data fork of the given file from the specified start byte to the specified end byte.

Usage

```
ReadFileRange(theFile, theStartByte, theEndByte)
```

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to read
theStartByte	*integer* the byte to start reading from (inclusive)
theEndByte	*integer* the byte to read to (inclusive)

Result

string	the contents of the file

Scripting additions required

Standard (Read/Write Commands)

ReadFileUntil( ) *File functions*

Reads the data fork of the given file from the specified start byte until the specified character is read.

Usage

```
ReadFileUntil(theFile, theStartByte, untilChar)
```

Parameters

theFile	*file path, file spec, or alias (or list thereof)* the file to read
theStartByte	*integer* the byte to start reading from (inclusive)
untilChar	*string* the character to read until (inclusive)

Result

string	the contents of the file

Scripting additions required

Standard (Read/Write Commands)

SetCreatorType( ) *File functions*

Sets the creator type of the supplied file (or list of files) to the given type.

Usage

```
SetCreatorType(theFileList, theCreatorCode)
```

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to set the creator type for
theCreatorCode	*string* the four-character creator code to apply

Result

nothing

Scripting additions required

Standard (File Commands)

SetEOF( ) *File functions*

Sets the end of file marker in the supplied file to the specified byte.

Usage

SetEOF(theFile, theByte)

Parameters

theFile	*file path, file spec, or alias* the file to set the end of file marker of
theByte	*integer* the byte marking the end of the file

Result

integer	new offset to the end of file marker

Scripting additions required

Standard (Read/Write Commands)

SetFileAndCreatorType( ) *File functions*

Sets the file type and creator type of the supplied file (or list of files) to the given types.

Usage

SetFileAndCreatorType(theFileList, theFileTypeCode, theCreatorCode)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to set the file and creator types for
theFileTypeCode	*string* the four-character file type code to apply (this parameter is ignored if supplied as an empty list or string)
theCreatorCode	*string* the four-character creator code to apply (this parameter is ignored if supplied as an empty list or string)

Result

nothing

Scripting additions required

Standard (File Commands)

SetFileType( ) *File functions*

Sets the file type of the supplied file (or list of files) to the given type.

Usage

SetFileType(theFileList, theFileTypeCode)

Parameters

theFileList	*file path, file spec, or alias (or list thereof)* the file (or list of files) to set the file type for
theFileTypeCode	*string* the four-character file type code to apply

Result

nothing

Scripting additions required

Standard (File Commands)

VerifyFile( ) *File functions*

Returns whether or not the given file item exists.

Usage

VerifyFile(theFile)

Parameters

theFile	*file path, file spec, or alias* the file item (file, folder, package, or disk) to test for

Result

boolean	*true* if the file item exists; *false* if it does not

Scripting additions required

None

VerifyFileTypes( ) *File functions*

Returns whether or not the given file of the specified file and creator types exists.

Usage

```
VerifyFileTypes(theFile, theFileType, theCreatorType)
```

Parameters

theFile	*file path, file spec, or alias* the file to test for
theFileType	*string* the four-character file type to test for (this parameter is ignored if supplied as an empty list or string)
theCreatorType	*string* the four-character creator type to test for (this parameter is ignored if supplied as an empty list or string)

Result

boolean	*true* if the file exists; *false* if it does not

Scripting additions required

Standard (File Commands)

WriteToFile( ) *File functions*

Writes the given data to the beginning of the specified file.

Usage

```
WriteToFile(theFile, theData)
```

Parameters

theFile	*file path, file spec, or alias* the file to write to
theData	*anything* the data to write

Result

nothing

Scripting additions required

Standard (Read/Write Commands)

WriteToFileAtByte( ) *File functions*

Writes the given data to the specified file, starting at the specified byte.

Usage

WriteToFileAtByte(theFile, theData, theByte)

Parameters

theFile	*file path, file spec, or alias* the file to write to
theData	*anything* the data to write
theByte	*integer* the byte to start writing to (inclusive)

Result

nothing

Scripting additions required

Standard (Read/Write Commands)

WriteToFileAtEnd( ) *File functions*

Writes the given data to the end of the specified file.

Usage

WriteToFileAtEnd(theFile, theData)

Parameters

theFile	*file path, file spec, or alias* the file to write to
theData	*anything* the data to write

Result

nothing

Scripting additions required

Standard (Read/Write Commands)

CheckTicks( ) *General functions*

Returns the number of ticks elapsed since *StartTicks( )* was called, without resetting the stored ticks counter. See also *ResetTicks( )*, *StartTicks( )*, and *StopTicks( )*.

Usage

CheckTicks( )

Parameters

none

Result

integer the number of elapsed ticks

Scripting additions required

Jon's Commands (not required if used in a FaceSpan project)

CheckTimer( ) *General functions*

Returns the number of seconds elapsed since *StartTimer( )* was called, without resetting the stored time counter. See also *ResetTimer( )*, *StartTimer( )*, and *StopTimer( )*.

Usage

CheckTimer( )

Parameters

none

Result

integer the number of elapsed seconds

Scripting additions required

Standard (Current Date)

ChooseFile( ) *General functions*

Displays the standard *choose file* dialog box with the given prompt, showing the given type(s).

Usage

```
ChooseFile(thePrompt, theTypeList)
```

Parameters

thePrompt	*string* the user prompt to display; defaults to "Choose a file." if supplied with an empty string or list; prompts longer than 256 characters are truncated.
theTypeList	*string (or list of string)* the four-character file type code(s) to display; lists of more than four codes are truncated to four codes.

Result

alias	the chosen file

Scripting additions required

Standard (Choose File)

ChooseFileOverProcess( ) *General functions*

Displays the standard *choose file* dialog box with the given prompt, showing the given type(s), over the given process. (Note: your code must still activate the process if desired and handle any timeouts.)

Usage

```
ChooseFileOverProcess(theProcessName, thePrompt, theTypeList)
```

Parameters

theProcessName	*string* the name of the process to display over
thePrompt	*string* the user prompt to display; defaults to "Choose a file." if supplied with an empty string or list; prompts longer than 256 characters are truncated.
theTypeList	*string (or list of string)* the four-character file type code(s) to display; lists of more than four codes are truncated to four codes.

Result

alias	the chosen file

Scripting additions required

Standard (Choose File)

ChooseFolder( ) *General functions*

Displays the standard *choose folder* dialog box with the given prompt.

Usage

```
ChooseFolder(thePrompt)
```

Parameters

thePrompt	*string* the user prompt to display; defaults to "Choose a folder." if supplied with an empty string or list; prompts longer than 256 characters are truncated.

Result

alias	the chosen folder

Scripting additions required

Standard (Choose File)

ChooseFolderOverProcess( ) *General functions*

Displays the standard *choose folder* dialog box with the given prompt over the given process. (Note: your code must still activate the process if desired and handle any timeouts.)

Usage

```
ChooseFolderOverProcess(theProcessName, thePrompt)
```

Parameters

theProcessName	*string* the name of the process to display over
thePrompt	*string* the user prompt to display; defaults to "Choose a folder." if supplied with an empty string or list; prompts longer than 256 characters are truncated.

Result

alias	the chosen folder

Scripting additions required

Standard (Choose File)

ChooseFileName( ) *General functions*

Displays the standard *choose file name* dialog box with the given prompt and the given default name.

Usage

```
ChooseFileName(thePrompt, theDefaultName)
```

Parameters

thePrompt	*string* the user prompt to display; defaults to "Save as:" if supplied with an empty string or list; prompts longer than 256 characters are truncated.
theDefaultName	*string* the filename to appear in the name box; defaults to nothing if supplied with an empty string or list; names longer than 31 characters are truncated.

Result

file spec	file spec for the file

Scripting additions required

Standard (New File)

ChooseFileNameOverProcess( ) *General functions*

Displays the standard *choose file name* dialog box with the given prompt and the given default name, over the given process. (Note: your code must still activate the process if desired and handle any timeouts.)

Usage

```
ChooseFileNameOverProcess(theProcessName, thePrompt, theDefaultName)
```

Parameters

theProcessName	*string* the name of the process to display over
thePrompt	*string* the user prompt to display; defaults to "Save as:" if supplied with an empty string or list; prompts longer than 256 characters are truncated.
theDefaultName	*string* the filename to appear in the name box; defaults to nothing if supplied with an empty string or list; names longer than 31 characters are truncated.

Result

file spec	file spec for the file

Scripting additions required

Standard (New File)

ChooseNewFile( ) *General functions*

Displays the standard *new file* dialog box with the given prompt and the given default name.

Usage

```
ChooseNewFile(thePrompt, theDefaultName)
```

Parameters

thePrompt	*string* the user prompt to display; defaults to "Save as:" if supplied with an empty string or list; prompts longer than 256 characters are truncated.
theDefaultName	*string* the filename to appear in the name box; defaults to nothing if supplied with an empty string or list; names longer than 31 characters are truncated.

Result

file spec	file spec for the file

Scripting additions required

Standard (New File)

ChooseNewFileOverProcess( ) *General functions*

Displays the standard *new file* dialog box with the given prompt and the given default name, over the given process. (Note: your code must still activate the process if desired and handle any timeouts.)

Usage

```
ChooseNewFileOverProcess(theProcessName, thePrompt, theDefaultName)
```

Parameters

theProcessName	*string* the name of the process to display over
thePrompt	*string* the user prompt to display; defaults to "Save as:" if supplied with an empty string or list; prompts longer than 256 characters are truncated.
theDefaultName	*string* the filename to appear in the name box; defaults to nothing if supplied with an empty string or list; names longer than 31 characters are truncated.

Result

file spec	file spec for the file

Scripting additions required

Standard (New File)

DisplayDialog( ) *General functions*

Displays a standard modal dialog box with the given options.

Usage

DisplayDialog(theMsg, theButtonList, theDefaultButton, theIconID, doBeep)

Parameters

theMsg	*string* the message text to display; messages longer than 256 characters are truncated
theButtonList	*string or list of string* the labels for the button(s); defaults to {"Cancel", "OK"} if supplied with an empty string or list; lists longer than 3 items are truncated
theDefaultButton	*integer or string* the index or name of the button to make the default button; defaults to none if supplied with an empty string or list
theIconID	*integer or constant* the resource ID or name constant (i.e. *stop*, *note* or *caution*) of the icon to display; defaults to none if supplied with an empty string or list
doBeep	*boolean* whether to precede the dialog box with a system beep; *true* to beep, *false* to not

Result

string	name of the button returned

Scripting additions required

Standard (Beep, Display Dialog)

DisplayDialogOverProcess( ) *General functions*

Displays a standard modal dialog box over the given process with the given options.
(Note: your code must still activate the process if desired and handle any timeouts.)

Usage

```
DisplayDialogOverProcess(theProcessName, theMsg, theButtonList,
                         theDefaultButton, theIconID, doBeep)
```

Parameters

theProcessName	*string* the name of the process to display over
theMsg	*string* the message text to display; messages longer than 256 characters are truncated
theButtonList	*string or list of string* the labels for the button(s); defaults to {"Cancel", "OK"} if supplied with an empty string or list; lists longer than 3 items are truncated
theDefaultButton	*integer or string* the index or name of the button to make the default button; defaults to none if supplied with an empty string or list
theIconID	*integer or constant* the resource ID or name constant (i.e. *stop*, *note* or *caution*) of the icon to display; defaults to none if supplied with an empty string or list
doBeep	*boolean* whether to precede the dialog box with a system beep; *true* to beep, *false* to not

Result

string	name of the button returned

Scripting additions required

Standard (Beep, Display Dialog)

DisplayInputDialog( ) *General functions*

Displays a standard input modal dialog box with the given options.

Usage

```
DisplayInputDialog(theMsg, theDefaultAnswer, theButtonList,
                   theDefaultButton, theIconID, doBeep)
```

Parameters

theMsg	*string* the message text to display; messages longer than 256 characters are truncated
theDefaultAnswer	*string* the default answer to appear in the input box; defaults to nothing if supplied with an empty string or list; strings longer than 255 characters are truncated
theButtonList	*string or list of string* the labels for the button(s); defaults to {"Cancel", "OK"} if supplied with an empty string or list; lists longer than 3 items are truncated
theDefaultButton	*integer or string* the index or name of the button to make the default button; defaults to none if supplied with an empty string or list
theIconID	*integer or constant* the resource ID or name constant (i.e. *stop*, *note* or *caution*) of the icon to display; defaults to none if supplied with an empty string or list
doBeep	*boolean* whether to precede the dialog box with a system beep; *true* to beep, *false* to not

Result

record	record of {button returned: "*name of button*", text returned: "*text*"}

Scripting additions required

Standard (Beep, Display Dialog)

DisplayInputDialogOverProcess( ) *General functions*

Displays a standard input modal dialog box over the given process with the given options. (Note: your code must still activate the process if desired and handle any timeouts.)

Usage

```
DisplayInputDialogOverProcess(theProcessName, theMsg, theDefaultAnswer,
                              theButtonList, theDefaultButton, theIconID, doBeep)
```

Parameters

theProcessName	*string* the name of the process to display over
theMsg	*string* the message text to display; messages longer than 256 characters are truncated
theDefaultAnswer	*string* the default answer to appear in the input box; defaults to nothing if supplied with an empty string or list; strings longer than 255 characters are truncated
theButtonList	*string or list of string* the labels for the button(s); defaults to {“Cancel”, “OK”} if supplied with an empty string or list; lists longer than 3 items are truncated
theDefaultButton	*integer or string* the index or name of the button to make the default button; defaults to none if supplied with an empty string or list
theIconID	*integer or constant* the resource ID or name constant (i.e. *stop*, *note* or *caution*) of the icon to display; defaults to none if supplied with an empty string or list
doBeep	*boolean* whether to precede the dialog box with a system beep; *true* to beep, *false* to not

Result

record — record of {button returned: "*name of button*", text returned: "*text*"}

Scripting additions required

Standard (Beep, Display Dialog)

GetClass( ) *General functions*

Returns the class of the given item (or list of items).

Usage

GetClass(theItems)

Parameters

theItems	*anything* the item (or list of items) to return class of

Result

constant	the class of each supplied item

Scripting additions required

None

GetClipboard( ) *General functions*

Returns the data currently on the clipboard. See also *SetClipboard()*, *StoreClipboard()*, and *RestoreClipboard()*.

Usage

GetClipboard( )

Parameters

none

Result

anything	the clipboard data

Scripting additions required

None

GetDelims( ) *General functions*

Returns the current value of AppleScript's text item delimiters. See also *SetDelims( )*, *ResetDelims( )*, and *SwapDelims( )*.

Usage

GetDelims( )

Parameters

none

Result

list the list of current text item delimiters

Scripting additions required

None

ResetDelims( ) *General functions*

Resets AppleScript's text item delimiters to the value previously stored by *SetDelims( )*. See also *GetDelims( )*, *SetDelims( )*, and *SwapDelims( )*.

Usage

ResetDelims( )

Parameters

none

Result

nothing

Scripting additions required

None

ResetTicks( ) *General functions*

Returns the number of ticks elapsed since *StartTicks( )* was called, and resets the stored ticks counter. See also *CheckTicks( )*, *StartTicks( )*, and *StopTicks( )*.

Usage

ResetTicks( )

Parameters

none

Result

integer the number of elapsed ticks

Scripting additions required

Jon's Commands (not required if used in a FaceSpan project)

ResetTimer( ) *General functions*

Returns the number of seconds elapsed since *StartTimer( )* was called, and resets the stored time counter. See also *CheckTimer( )*, *StartTimer( )*, and *StopTimer( )*.

Usage

ResetTimer( )

Parameters

none

Result

integer the number of elapsed seconds

Scripting additions required

Standard (Current Date)

RestoreClipboard( ) *General functions*

Restores the data to the clipboard which was previously stored by StoreClipboard().
See also *GetClipboard()*, *SetClipboard()*, and *StoreClipboard()*.

Usage

RestoreClipboard( )

Parameters

none

Result

nothing

Scripting additions required

None

SetClipboard( ) *General functions*

Sets the clipboard to the supplied data. See also *GetClipboard()*, *StoreClipboard()*, and *RestoreClipboard()*.

Usage

SetClipboard(theData)

Parameters

theData	*anything* the data to place on the clipboard

Result

nothing

Scripting additions required

None

SetDelims( ) *General functions*

Sets AppleScript's text item delimiters to the supplied delimiter, storing the previous delimiters. See also *GetDelims( )*, *ResetDelims( )*, and *SwapDelims( )*.

Usage

```
SetDelims(theDelimiter)
```

Parameters

theDelimiter	*string or list of string* the new text item delimiter (or list thereof)

Result

nothing

Scripting additions required

None

StartTicks( ) *General functions*

Sets the stored tick counter to the current tick value. See also *CheckTicks( )*, *ResetTicks( )*, and *StopTicks( )*.

Usage

```
StartTicks( )
```

Parameters

none

Result

nothing

Scripting additions required

Jon's Commands (not required if used in a FaceSpan project)

StartTimer( ) *General functions*

Sets the stored time counter to the current time value. See also *CheckTimer( )*, *ResetTimer( )*, and *StopTimer( )*.

Usage

StartTimer( )

Parameters

none

Result

nothing

Scripting additions required

Standard (Current Date)

StopTicks( ) *General functions*

Returns the number of ticks elapsed since the tick count was started by *StartTicks( )*. See also *CheckTicks( )*, *StartTicks( )*, and *ResetTicks( )*.

Usage

StopTicks( )

Parameters

none

Result

integer the number of elapsed ticks

Scripting additions required

Jon's Commands (not required if used in a FaceSpan project)

StopTimer( ) *General functions*

Returns the number of seconds elapsed since the time count was started by *StartTimer( )*.
See also *CheckTimer( )*, *StartTimer( )*, and *ResetTimer( )*.

Usage

StopTimer( )

Parameters

none

Result

integer the number of elapsed seconds

Scripting additions required

Standard (Current Date)

StoreClipboard( ) *General functions*

Stores the data currently on the clipboard (for restoration later).
See also *GetClipboard( )*, *SetClipboard( )*, and *RestoreClipboard( )*.

Usage

StoreClipboard( )

Parameters

none

Result

nothing

Scripting additions required

None

SwapDelims( ) *General functions*

Sets AppleScript's text item delimiters to the supplied delimiter and returns the previous delimiters. Note: *SwapDelims( )* does not modify the stored text item delimiters property. See also *GetDelims( )*, *ResetDelims( )*, and *SetDelims( )*.

Usage

SwapDelims(theNewDelimiter)

Parameters

theNewDelimiter	*string or list of string* the new text item delimiter (or list thereof)

Result

list of string	the previous text item delimiters

Scripting additions required

None

DeList( ) *List functions*

Returns item 1 of a single-item list; otherwise returns the original list. See also *ListWrap( )*.

Usage

DeList(theList) or DL(theList)

Parameters

theList	*list* the single-item list to turn into an item

Result

anything	the item from the list

Scripting additions required

None

ExcludeFromList( ) *List functions*

Excludes the given item or items from the supplied list. See also *RemoveFromList( )*.

Usage

ExcludeFromList(theList, theItemsToExclude)

Parameters

theList	*list* the list of items to test
theItemsToExclude	*anything* the item (or list of items) to exclude from the list

Result

list	list of non-excluded items

Scripting additions required

None

FlattenList( ) *List functions*

Returns a single-level (flat) list from a list of nested lists.

Usage

FlattenList(theList)

Parameters

theList	*list* the list of nested lists to flatten

Result

list	the flattened list

Scripting additions required

None

GetOffsetInList( ) *List functions*

Returns the offset to the given item in the supplied list.

Usage

GetOffsetInList(theItem, theList)

Parameters

theItem	*anything* the item to find
theList	*list* the list to search in

Result

integer	the offset (index) to the first matching item in the list; returns 0 if not found

Scripting additions required

None

GetTextItems( ) *List functions*

Breaks the supplied string into a list of text items wherever the given delimiter occurs.

Usage

```
GetTextItems(theText, theDelimiter)
```

Parameters

theText	*string* the string to get text items from
theDelimiter	*string* the delimiter to use to separate the text items

Result

list of string	the list of text items

Scripting additions required

None

GetUniqueListItems( ) *List functions*

Returns a list of the unique items from the supplied list. Note: for faster processing of lists of text items, use *GetUniqueTextListItems()*.

Usage

```
GetUniqueListItems(theList)
```

Parameters

theList	*list* the list to get unique items from

Result

list	the list of unique items

Scripting additions required

None

GetUniqueTextListItems( ) *List functions*

Returns a list of the unique text items from the supplied list of text items.

Usage

```
GetUniqueTextListItems(theTextList)
```

Parameters

theTextList	*list of string* the list to get unique text items from

Result

list of string	list of unique text items

Scripting additions required

None

InvertMatrix( ) *List functions*

Inverts the *x* and *y* axes of the given matrix (nested list).

Usage

```
InvertMatrix(theList)
```

Parameters

theList	*list of list* list of nested lists to invert

Result

list	the inverted matrix

Scripting additions required

None

ListWrap( ) *List functions*

Turns the given item into a single-item list containing the given item (if it is not already a list). See also *DeList( )*.

Usage

ListWrap(theItem) or LW(theItem)

Parameters

theItem	*anything* the item to turn into a list

Result

list	the item as a list

Scripting additions required

None

MergeRecords( ) *List functions*

Merges the properties of the supplied records. If a record is supplied then only that copy of the record is changed; if a reference to a record is supplied then the original record is changed.

Usage

MergeRecords(theSourceRecord, theRecordToAdd, doReplace)

Parameters

theSourceRecord	*record (or reference to a record)* the source record (or reference to)
theRecordToAdd	*record (or reference to a record)* the record (or reference to record) of properties to add to the source
doReplace	*boolean* whether or not to replace any duplicate properties in the source record

Result

record	the combined record of properties

Scripting additions required

None

RemoveFromList( ) *List functions*

Removes the given item number (index) from the supplied list. See also *ExcludeFromList( )*.

Usage

```
RemoveFromList(theList, theItemNumber)
```

Parameters

theList	*list* the list of items
theItemNumber	*integer* the index of the list item to remove

Result

list	amended list of items

Scripting additions required

None

SortList( ) *List functions*

Sorts the supplied list in ascending order. Note: to sort in descending order use the AppleScript *reverse* keyword (i.e. *reverse of SortList(theList)*).

Usage

```
SortList(theList)
```

Parameters

theList	*list of anything* list of items (strings, numbers, or dates) to sort

Result

list	the sorted list

Scripting additions required

None

AbsValue( ) *Math functions*

Returns the absolute value of the supplied number.

Usage

AbsValue(theNumber)

Parameters

theNumber	*number (integer or real)* number to get absolute value of

Result

number	the absolute value

Scripting additions required

None

AverageOf( ) *Math functions*

Returns the average (mean) of a supplied list of numbers. See also *MeanOf()*.

Usage

AverageOf(theNumberList)

Parameters

theNumberList	*list of numbers (integer or real)* the list of numbers to average

Result

real	the average value

Scripting additions required

None

BinToDec( ) *Math functions*

Converts the supplied binary string (or list of strings) into decimal integer(s).
See also *DecToBin( )*.

Usage

BinToDec(theBinaryStringList)

Parameters

theBinaryStringList	*string (or list of string)* the binary string(s) to convert

Result

integer	the decimal value (or list of values)

Scripting additions required

None

CosOf( ) *Math functions*

Returns the cosine of the supplied degree angle.

Usage

CosOf(theDegreeAngle)

Parameters

theDegreeAngle	*number (integer or real)* the degree angle to compute the cosine of

Result

real	the cosine value

Scripting additions required

None

DecToBin( ) *Math functions*

Converts the supplied positive decimal integer (or list of integers) into binary string(s).
See also *BinToDec( )*.

Usage

DecToBin(theDecimalNumberList)

Parameters

theDecimalNumberList	*integer (or list of integer)* the positive integer(s) to convert

Result

string	the binary value (or list of values)

Scripting additions required

None

DecToHex( ) *Math functions*

Converts the supplied positive decimal integer (or list of integers) into hexadecimal string(s).
See also *HexToDec( )*.

Usage

DecToHex(theDecimalNumberList)

Parameters

theDecimalNumberList	*integer (or list of integer)* the positive integer(s) to convert

Result

string	the hexadecimal value (or list of values)

Scripting additions required

None

DecToOct( ) *Math functions*

Converts the supplied positive decimal integer (or list of integers) into octal string(s).
See also *OctToDec( )*.

Usage

DecToOct(theDecimalNumberList)

Parameters

theDecimalNumberList	*integer (or list of integer)* the positive integer(s) to convert

Result

string	the octal value (or list of values)

Scripting additions required

None

DegToRad( ) *Math functions*

Converts the supplied degree angle into radians. See also *RadToDeg( )*.

Usage

DegToRad(theDegreeAngle)

Parameters

theDegreeAngle	*number (integer or real)* the degree angle to convert

Result

real	the radian measure of the angle

Scripting additions required

None

DivMod( ) *Math functions*

Returns the dividend and the modulus of the supplied number when divided by the supplied value.

Usage

DivMod(theNumber, theDivisor)

Parameters

theNumber	*number (integer or real)* the number to divide
theDivisor	*number (integer or real)* the number to divide by

Result

list	list of {dividend, modulus} of the division

Scripting additions required

None

Factorial( ) *Math functions*

Returns the factorial of the supplied positive integer.

Usage

Factorial(theInteger)

Parameters

theInteger	*positive integer* the integer to compute factorial of

Result

integer	the factorial result

Scripting additions required

None

HexToDec( ) *Math functions*

Converts the supplied hexadecimal string (or list of strings) into decimal integer(s).
See also *DecToHex( )*.

Usage

HexToDec(theHexStringList)

Parameters

theHexStringList *string (or list of string)*
the hexadecimal string(s) to convert

Result

integer the decimal value (or list of values)

Scripting additions required

None

IsNumberEven( ) *Math functions*

Returns whether or not the supplied number is even. See also *IsNumberOdd( )*.

Usage

IsNumberEven(theNumber)

Parameters

theNumber *number (integer or real)*
the number to test

Result

boolean *true* if the number is even; *false* if it is not

Scripting additions required

None

IsNumberOdd( ) *Math functions*

Returns whether or not the supplied number is odd. See also *IsNumberEven( )*.

Usage

IsNumberOdd(theNumber)

Parameters

theNumber	*number (integer or real)* the number to test

Result

boolean	*true* if the number is odd; *false* if it is not

Scripting additions required

None

IsNumberPrime( ) *Math functions*

Returns whether or not the supplied number is prime.

Usage

IsNumberPrime(theNumber)

Parameters

theNumber	*positive integer* the number to test

Result

boolean	*true* if the number is prime; *false* if it is not

Scripting additions required

None

LogBase10( ) *Math functions*

Returns the base 10 logarithm of the supplied number (or list of numbers).

Usage

LogBase10(theNumberList)

Parameters

theNumberList	*positive number, or list of numbers (integer or real)* number or list of numbers to get logarithm of

Result

real	the base 10 logarithm of the number (or list thereof)

Scripting additions required

None

LogBaseE( ) *Math functions*

Returns the base *e* (natural) logarithm of the supplied number (or list of numbers).

Usage

LogBaseE(theNumberList)

Parameters

theNumberList	*positive number, or list of numbers (integer or real)* number or list of numbers to get logarithm of

Result

real	the base *e* logarithm of the number (or list thereof)

Scripting additions required

None

LogBaseN() *Math functions*

Returns the logarithm of the specified base of the supplied number (or list of numbers).

Usage

LogBaseN(theNumberList, theBase)

Parameters

theNumberList	*positive number, or list of numbers (integer or real)* number or list of numbers to get logarithm of
theBase	*positive number (integer or real)* the base to compute logarithm from

Result

real	the logarithm of the number (or list thereof)

Scripting additions required

None

MeanOf() *Math functions*

Returns the mean (average) of a supplied list of numbers. See also *AverageOf()*.

Usage

MeanOf(theNumberList)

Parameters

theNumberList	*list of numbers (integer or real)* list of numbers to average

Result

real	the mean value

Scripting additions required

None

OctToDec( ) *Math functions*

Converts the supplied octal string (or list of strings) into decimal integer(s).
See also *DecToOct( )*.

Usage

OctToDec(theOctalStringList)

Parameters

theOctalStringList	*string (or list of string)* the octal string(s) to convert

Result

integer	the decimal value (or list of values)

Scripting additions required

None

ProductOf( ) *Math functions*

Multiplies the given list of numbers and returns their product.

Usage

ProductOf(theNumberList)

Parameters

theNumberList	*list of numbers (integer or real)* the list of numbers to multiply

Result

number	the product of the supplied numbers

Scripting additions required

None

RadToDeg( ) *Math functions*

Converts the supplied radian angle into degrees. See also *DegToRad( )*.

Usage

```
RadToDeg(theRadianAngle)
```

Parameters

theRadianAngle	*number (integer or real)* the radian angle to convert

Result

real	the degree measure of the angle

Scripting additions required

None

RoundNumber( ) *Math functions*

Rounds the supplied number (or list of numbers) to the specified number of decimal places. See also *FormatNumber( )*.

Usage

```
RoundNumber(theNumberList, thePrecision)
```

Parameters

theNumberList	*number (or list of numbers)* the number (or list of numbers) to round
thePrecision	*integer* the number of decimal places to round to; a negative number will round from the left of the decimal

Result

number	the rounded number (or list of numbers)

Scripting additions required

None

SinOf( ) *Math functions*

Returns the sine of the supplied degree angle.

Usage

SinOf(theDegreeAngle)

Parameters

theDegreeAngle	*number (integer or real)* the degree angle to compute the sine of

Result

real	the sine value

Scripting additions required

None

SumOf( ) *Math functions*

Adds the given list of numbers and returns their sum.

Usage

SumOf(theNumberList)

Parameters

theNumberList	*list of numbers* list of numbers to add

Result

number	the sum of the supplied numbers

Scripting additions required

None

TanOf( ) *Math functions*

Returns the tangent of the supplied degree angle.

Usage

TanOf(theDegreeAngle)

Parameters

theDegreeAngle	*number (integer or real)* the degree angle to compute the tangent of

Result

real	the tangent value

Scripting additions required

None

ActivateApp( ) *Process functions*

Sends the activate event to the given application. See also *LaunchApp( )* and *QuitApp( )*.

Usage

ActivateApp(theApp)

Parameters

theApp	*string, file spec, or alias* the creator type, file, or name (if it is running) of the application to activate

Result

nothing

Scripting additions required

Standard (File Commands)

ActivateProcess( ) *Process functions*

Activates (brings to front) the given process. See also *DeactivateProcess( )*.

Usage

ActivateProcess(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to activate

Result

nothing

Scripting additions required

None

DeactivateProcess( ) *Process functions*

Deactivates (removes from front) the given process. See also *ActivateProcess( )*.

Usage

DeactivateProcess(theProcess)

Parameters

theProcess — *string, file spec, or alias*
the creator type, file, or name of the process to deactivate

Result

nothing

Scripting additions required

None

DoesAppExist( ) *Process functions*

Returns whether or not the given application exists on the local machine.

Usage

DoesAppExist(theApp)

Parameters

theApp — *string, file spec, or alias*
the creator type, file, or name (if it is running) of the application to test

Result

boolean — *true* if the given application exists; *false* if it does not

Scripting additions required

Standard (File Commands)

GetAppCreator( ) *Process functions*

Returns the creator type of the given application. See also *GetAppFile( )* and *GetAppName( )*.

Usage

```
GetAppCreator(theApp)
```

Parameters

theApp	*string, file spec, or alias* the creator type, file, or name (if it is running) of the application to get the creator type of

Result

string	creator type of the given application

Scripting additions required

Standard (File Commands)

GetAppFile( ) *Process functions*

Returns the file path of the given application. See also *GetAppCreator( )* and *GetAppName( )*.

Usage

```
GetAppFile(theApp)
```

Parameters

theApp	*string, file spec, or alias* the creator type, file, or name (if it is running) of the application to get the file path of

Result

string	file path of the given application

Scripting additions required

Standard (File Commands)

GetAppName( ) *Process functions*

Returns the name of the given application. See also *GetAppCreator( )* and *GetAppFile( )*.

Usage

GetAppName(theApp)

Parameters

theApp	*string, file spec, or alias*	the creator type, file, or name (if it is running) of the application to get the name of

Result

string	name of the given application

Scripting additions required

Standard (File Commands)

GetAppRecord( ) *Process functions*

Returns a collection of properties for the given application. See also *InitAppRecord( )*.

Usage

GetAppRecord(theApp)

Parameters

theApp	*string, file spec, or alias*	the creator type, file, or name (if it is running) of the application to get properties of

Result

record — record of properties for the given application:

appExists	*boolean*	whether or not the application exists
creator	*string*	the creator type of the application
file	*string*	the file path of the application
name	*string*	the name of the application (or the process name if it is already running)
alreadyRunning	*boolean*	whether or not the application is already running

Scripting additions required

Standard (File Commands)

GetProcessCreator( ) *Process functions*

Returns the creator type of the given process. See also *GetProcessFile()* and *GetProcessName()*.

Usage

```
GetProcessCreator(theProcess)
```

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to get the creator type of

Result

string	creator type of the given process

Scripting additions required

None

GetProcessFile( ) *Process functions*

Returns the file path for the given process. See also *GetProcessCreator()* and *GetProcessName()*.

Usage

```
GetProcessFile(theProcess)
```

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to get the file path of

Result

string	file path of the given process

Scripting additions required

None

GetProcessName( ) *Process functions*

Returns the name for the given process. See also *GetProcessCreator( )* and *GetProcessFile( )*.

Usage

GetProcessName(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to get the name of

Result

string	name of the given process

Scripting additions required

None

HideProcess( ) *Process functions*

Hides the given process. See also *UnhideProcess( )*.

Usage

HideProcess(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to hide

Result

nothing

Scripting additions required

None

InitAppRecord( ) *Process functions*

Initializes the values within a pre-defined application record. See also *GetAppRecord()*.

Usage

InitAppRecord(theAppRecordRef)

Parameters

theAppRecordRef *record reference*
the record to populate with information

Recognized symbols:

appExists	*boolean*	whether or not the application exists
creator	*string*	the creator type of the application
file	*string*	the file path of the application
name	*string*	the name of the application (or the process name if it is already running)
alreadyRunning	*boolean*	whether or not the application is already running
errMsg	*string*	the error message to return if the application is not found

Note: the record to be initialized does not need to contain all of these properties, but must at least contain a *creator*, *file*, or *name* property.

Result

nothing

Scripting additions required

Standard (File Commands)

IsProcessHidden( ) *Process functions*

Returns whether or not the given process is hidden.

Usage

IsProcessHidden(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to test

Result

boolean	*true* if the given process is hidden; *false* if it is not

Scripting additions required

None

IsProcessInFront( ) *Process functions*

Returns whether or not the given process is in front.

Usage

IsProcessInFront(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to test

Result

boolean	*true* if the given process is in front; *false* if it is not

Scripting additions required

None

IsProcessRunning( ) *Process functions*

Returns whether or not the given process is running.

Usage

IsProcessRunning(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to test

Result

boolean	*true* if the given process is running; *false* if it is not

Scripting additions required

None

LaunchApp( ) *Process functions*

Sends the launch event to the given application. See also *ActivateApp()* and *QuitApp()*.

Usage

LaunchApp(theApp)

Parameters

theApp	*string, file spec, or alias* the creator type, file, or name (if it is running) of the application to launch

Result

nothing

Scripting additions required

Standard (File Commands)

QuitApp( ) *Process functions*

Sends the quit event to the given application. See also *ActivateApp( )* and *LaunchApp( )*.

Usage

QuitApp(theApp)

Parameters

theApp	*string, file spec, or alias* the creator type, file, or name (if it is running) of the application to quit

Result

nothing

Scripting additions required

Standard (File Commands)

QuitProcess( ) *Process functions*

Sends the quit event to the given process.

Usage

QuitProcess(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to quit

Result

nothing

Scripting additions required

None

UnhideProcess( ) *Process functions*

Unhides (makes visible) the given process. See also *HideProcess( )*.

Usage

UnhideProcess(theProcess)

Parameters

theProcess	*string, file spec, or alias* the creator type, file, or name of the process to make visible

Result

nothing

Scripting additions required

None

FormatDate( ) *String functions*

Returns a date string for each of the supplied date(s), formatted according to the given template.

Usage

FormatDate(theDateList, theDateTemplate)

Parameters

theDateList *date (or list of dates)*
the date (or list of dates) to format;
defaults to current date if supplied as an empty list or string

theDateTemplate *string*
the template to follow for date formatting;
defaults to a short form of the system's current format if an empty string is supplied

Template symbols:

d	number of day
dd	number of day (with leading zero if required)
Weekday or weekday	name of day (in English)
Day or day	truncated (3 character) name of weekday
m	number of month
mm	number of month (with leading zero if required)
Month or month	name of month (in English)
Mon or mon	truncated (3 character) name of month
yy	number of last 2 digits of year
yyyy or Year or year	number of all four digits of year

Note: all other text is returned unchanged.

Result

string the formatted date string (or list of date strings)

Examples

```
set theDate to date "Friday, 1 January 1904 12:00:00 AM"
```

FormatDate(theDate, "mm-dd-yyyy")	*returns:*	"01-01-1904"
FormatDate(theDate, "Weekday, d Month yyyy")	*returns:*	"Friday, 1 January 1904"
FormatDate(theDate, "circa month year")	*returns:*	"circa January 1904"

Scripting additions required

Standard (Current Date)

FormatTime( ) *String functions*

Returns a time string for each of the supplied date(s) or integer(s), formatted according to the given template.

Usage

FormatTime(theValueList, theTimeTemplate)

Parameters

theValueList — *date or integer (or list thereof)*
the date or integer (or list thereof) to format
defaults to current date if supplied as an empty list or string

theTimeTemplate — *string*
the template to follow for formatting;
defaults to the system's current time format if an empty string is supplied (either "24hh:mm:ss" or "12hh:mm:ss AM")

Template symbols:

24hh, 24HH	number of hours in 24-hour format (with leading zero if less than 10)
24h, 24H	number of hours in 24-hour format
12hh, 12HH	number of hours in 12-hour format (with leading zero if less than 10)
12h, 12H	number of hours in 12-hour format
hh, HH	number of hours in current hour format (with leading zero if less than 10)
h, H	number of hours in current hour format
mm, MM	number of minutes (with leading zero if less than 10)
m, M	number of minutes
ss, SS	number of seconds (with leading zero if less than 10)
s, S	number of seconds
am, pm	display lower case meridian label (in 12-hour format)
AM, PM	display upper case meridian label (in 12-hour format)

Note: all other text is returned unchanged.

Result

string — the formatted time string (or list of time strings)

Examples

```
set theDate to date "Friday, 1 January 1904 5:00:00 PM"
set theSeconds to 3600

FormatTime(theDate, "24hh:mm:ss")        returns: "17:00:00"
FormatTime(theSeconds, "h:mm")           returns: "1:00"
```

Scripting additions required

Standard (Current Date)

FormatNumber( ) *String functions*

Returns a formatted number string for each of the supplied numbers, rounded to the given decimal place (with added zeros if required). The default thousands separator and decimal point can be changed by calling *SetSeparatorChar()* and *SetDecimalChar()*, respectively, at runtime.

Usage

```
FormatNumber(theNumberList, thePrecision)
```

Parameters

theNumberList	*number (or list of numbers)* the number (or list of numbers) to format
thePrecision	*integer* the number of decimal places to round to; a negative number will round from the left of the decimal

Result

string	the formatted number string (or list of number strings)

Scripting additions required

None

GetASCIIchars( ) *String functions*

Returns a contiguous string of character(s) represented by the supplied ASCII number (or list of numbers).

Usage

```
GetASCIIchars(theNumberList)
```

Parameters

theNumberList	*integer (or list of integer)* the number (or list of numbers) to get ASCII characters for

Result

string	the ASCII characters

Scripting additions required

None

GetASCIInumbers( ) *String functions*

Returns a list of the ASCII number(s) representing the characters making up the supplied string.

Usage

GetASCIInumbers(theString)

Parameters

theString	*string* the string of characters to get ASCII numbers for

Result

list of integer	the list of ASCII numbers

Scripting additions required

None

GetOffsetInString( ) *String functions*

Returns the character offset to the supplied item in the given string.

Usage

GetOffsetInString(theItem, theString)

Parameters

theItem	*string* the string to get offset of (case sensitive)
theString	*string* the string to search in

Result

integer	the offset to the first matching character in the string; returns 0 if not found

Scripting additions required

None

MakeLowerCase( ) *String functions*

Changes the case of all characters in the supplied string to lower case.

Usage

```
MakeLowerCase(theText)
```

Parameters

theText	*string* the string of text to convert to lower case

Result

string	the converted string

Scripting additions required

None

MakeProperCase( ) *String functions*

Changes the case of characters in the supplied string to proper case (initial capitals).

Usage

```
MakeProperCase(theText)
```

Parameters

theText	*string* the string of text to convert to proper case

Result

string	the converted string

Scripting additions required

None

MakeUpperCase( ) *String functions*

Changes the case of all characters in the supplied string to upper case.

Usage

MakeUpperCase(theText)

Parameters

theText	*string* the string of text to convert to upper case

Result

string	the converted string

Scripting additions required

None

ReplaceString( ) *String functions*

Replaces the given search string (or list of search strings) with the given replace string (or list of replace strings) in the supplied text.

Usage

ReplaceString(theString, theSearchItems, theReplaceItems)

Parameters

theString	*string* the string to perform the replace on
theSearchItems	*string (or list of string)* the string (or list of strings) to search for
theReplaceItems	*string (or list of string)* the string (or list of strings) to replace with

Result

string	the modified text

Scripting additions required

None

SetDecimalChar( ) *String functions*

Sets the stored decimal character (used by *FormatNumber( )*) to the given character.

Usage

SetDecimalChar(theCharacter)

Parameters

theCharacter *string*
the character to act as the decimal point;
default is the period character (".")

Result

nothing

Scripting additions required

None

SetSeparatorChar( ) *String functions*

Sets the stored thousands separator character (used by *FormatNumber( )*) to the given character.

Usage

SetSeparatorChar(theCharacter)

Parameters

theCharacter *string*
the character to act as the thousands separator;
default is the comma character (",")

Result

nothing

Scripting additions required

None

TextListToString( ) *String functions*

Combines the supplied text items to a single string, placing the given delimiter between items.

Usage

TextListToString(theTextList, theDelimiter)

Parameters

theTextList	*list of string* the list of text items to combine
theDelimiter	*string* the text to insert between each text item

Result

string	the combined string

Scripting additions required

None

TextToNumber( ) *String functions*

Converts the supplied formatted number string (or list of strings) into a number (or list of numbers).

Usage

TextToNumber(theTextList)

Parameters

theTextList	*string (or list of string)* the number text (or list thereof) to convert

Result

number	the numeric value (or list of values) of the string

Scripting additions required

None

TrimText( ) *String functions*

Removes any white space characters (space, option-space, tab, and return) from both ends of the supplied string.

Usage

TrimText(theText)

Parameters

theText	*string* the text to trim

Result

string	the trimmed text

Scripting additions required

None

TruncateString( ) *String functions*

Truncates the supplied string to be the given length of characters. If the given length is negative, the left side of the string is truncated. Strings shorter than the given length are not affected.

Usage

TruncateString(theString, theLength)

Parameters

theString	*string* the text to truncate
theLength	*integer* the number of characters to truncate to; supplying a negative integer will truncate the left side of the string; defaults to 31 if supplied as an empty list or string, or as 0

Result

string	the truncated string

Scripting additions required

None