Script Tools 1.3

Reference Guide

Script Tools 1.3-1 and Script Tools 1.3 Reference Guide

1571 Deep Cove Road
North Vancouver, B.C.
CANADA V7G - 1S4

Internet: alldritt@wimsey.com

NOTICE:

The Script Tools software and this document are provided AS IS. The author is not responsible for any damages caused either directly or indirectly by the Script Tools software.

Table Of Contents

Addition/Command Cross Reference 1

Introduction

The ScriptTools package contains a series of additions to AppleScript, Apple's new scripting language for the Macintosh. If you are trying to automate any type of activity with AppleScript then Script Tools is a must. Many of the features of ScriptTools allow you to do things which simply cannot be done with AppleScript alone.

About this guide

This guide provides reference material describing each of the Script Tools AppleScript additions.

This guide assumes you are already familiar with the Macintosh and have some experience with AppleScript. If you're unfamiliar with these skills, refer to the manuals that came with your computer and AppleScript.

What you need to get started

To use Script Tools, your Macintosh computer must be running system software version 7.0 or later; have at least 4 megabytes of memory; and have AppleScript 1.0 or later installed.

To use the Speech AppleScript addition you will require version 1.1.1 of Apple's Speech Manager software.

Using and copying Script Tools

Please feel free to distribute Script Tools to friends and colleagues. However, Script Tools may not appear as part of any promotional offer or commercial product without the author's expressed written

permission. Commercial re-distribution licenses are available through the author.

When distributing Script Tools, please distribute the entire package as you received it.

Installing Script Tools on your Macintosh

To install Script Tools, copy the contents of the Additions folder to the Scripting Additions folder within the Extensions folder in your System Folder.

Script Tools Examples

The Examples folder contains a series of example AppleScript scripts showing how to use the new commands provided by Script Tools.

Choose Folder Example
: This short example shows the Script Tools Choose Folder command in use.

Choose File In Prefs Folder
: This example shows how the use the Set Default Folder and Get Default Folder commands to control the starting folder presented by the Choose File command.

Choose New File Example
: This short example shows the Script Tools Choose New File command in use.

Choose Several Files Example
: This short example shows the Script Tools Choose Several Files command in use.

Choose Several Folders Example
: This short example shows the Script Tools Choose Several Folders command in use.

Backup Folders
: This script uses the Script Tools Choose Several Folders and Choose Folder command to identify a series of folders that are to be backed up and a folder where the backup is to be stored. The backup is performed using StuffIt Lite via AppleEvent commands.

Shutdown
: This example illustrates the use of the Script Tools Shutdown command.

File IO Example This example creates a text file and writes a short message to it using the Script Tools File IO commands.

File IO Example II
This example opens a text file and displays the contents of the file line by line.

Regular Expression Example
This example uses the Script Tools Regular Expression commands to modify the names of all the files in a folder (note the file names are not actually changed).

Regular Expression Example II
This example uses Regular Expression and File IO commands to read and parse a simple text file.

List Folders This example uses the Choose Several Folders and the File IO commands to produce a listing of the files stored in folders.

Check For Speech Mgr
This example uses the Get Gestalt command to check for the presents of the Speech Manager.

Quit All Applications
This example illustrates how to use the List Processes, Get Process and Get Current Process commands to quit all the non-essential applications running on your Macintosh.

All of these examples are stored as Script Editor text files with the exception of Folder Watcher and List Folders which are compiled AppleScript application. The examples stored as text files can be opened using the AppleScript Script Editor or any text editor which can read TEXT files. The Folder Watcher and List Folders scripts can only be viewed using the Script Editor.

Script Tools Libraries

The Libraries folder contains the following AppleScript libraries.

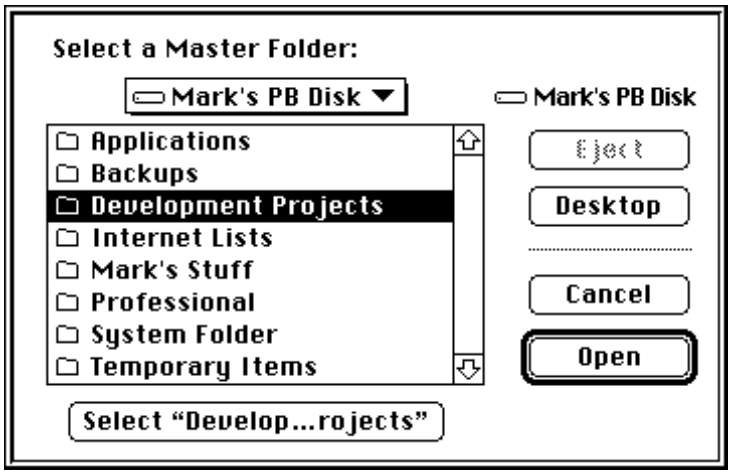
gestalt Selector Lib
This library defines the Gestalt selectors which are documented in Inside Macintosh volume VI.

Script Tools Additions

This chapter describes each of the AppleScript commands in the ScriptTools package. ScriptTools implements these new commands using AppleScript additions . AppleScript additions are a special type of software which adds new features to the AppleScript language.

Choose Folder

The Choose Folder command allows the user to choose a folder by displaying a dialog box like the one below.



Syntax

```
choose folder
    [ with prompt promptString ]
```

Parameters

promptString This parameter is a string which is displayed in the dialog box. If you omit the `with prompt` parameter, no prompt is displayed.

Result

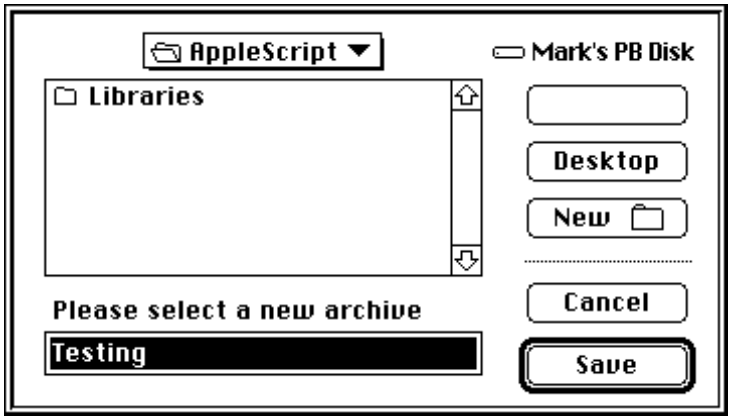
The result is an alias to the folder selected by the user.

Example

```
choose folder ¬
    with prompt "Please select a backup folder"
```

Choose New File

The Choose New File command presents the standard Macintosh new file selection dialog box.



Syntax

```
choose new file
    [ with prompt promptString ]
    [ default name nameString ]
```

Parameters

promptString This parameter is a string which is displayed in the dialog box. If this parameter is omitted the string "Save As:" is displayed.

nameString This parameter is a string which is offered as the default name for the new file. If this parameter is omitted no default name is presented.

Result

The result of the Choose New File is a record containing three values:

`filename returned`
: This value is a string representing the name of the new file.

`folder returned`
: This value is an alias to the folder where the new file is to be placed.

`replacing`
: This Boolean value indicates whether or not the new file replaces an existing file (TRUE = Yes, FALSE = No).

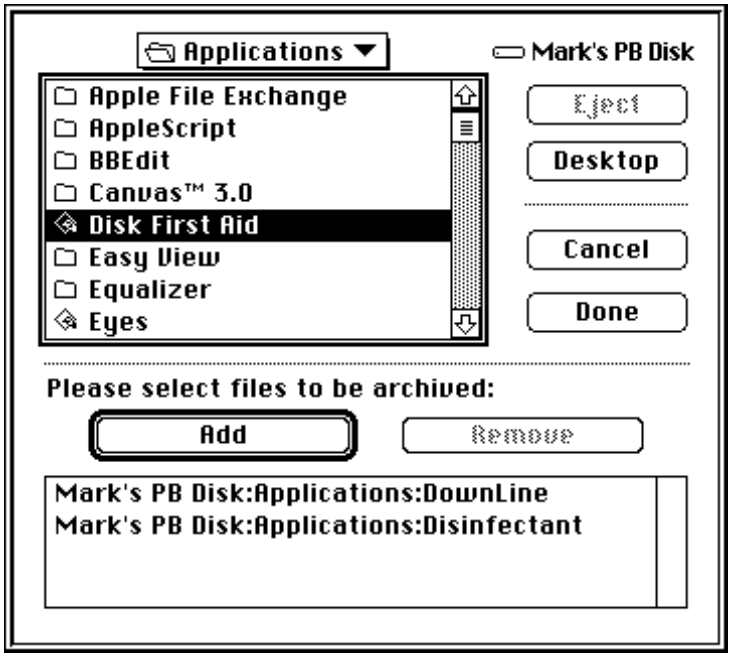
Example

```
-- Ask the user for a new file
set newFile to choose new file ¬
    with prompt "Select a new archive file:" ¬
    default name "Testing"

-- Show the result on the Script Editor result window
{ (folder returned of newFile as string), ¬
  (name returned of newFile) }
```

Choose Several Files

The Choose Several Files command presents a modified standard file selection dialog box allowing the user to choose several files at one time.



Syntax

```
choose several files
    [ with prompt promptString ]
    [ of type typeList ]
    [ starting with fileList ]
```

Parameters

promptString This parameter is a string which is displayed in the dialog box. If you omit the `with prompt` parameter, no prompt is displayed.

typeList This parameter is a list of strings specifying the file types of the files to be displayed in the dialog box. Each string is a four-character code for the file type, such as `"TEXT"`, `"APPL"`, `"PICT"` or `"PNTG"`. If you omit the `of type` parameter, all files are displayed. You may specify up to four file types.

fileList This parameter is a list of aliases referring to files which are to be displayed as already selected. If you omit the `starting with` parameter, the selected files list is left empty.

Result

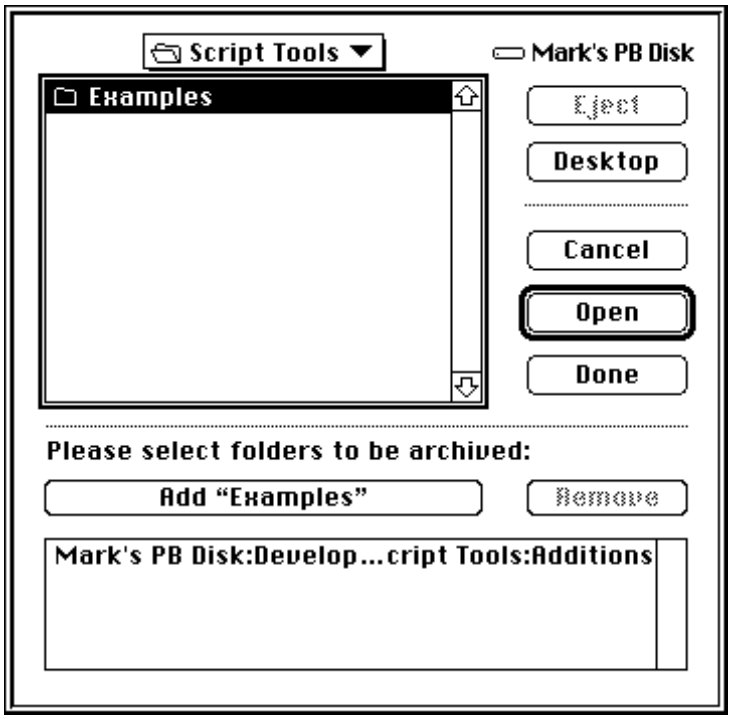
The result is a list of aliases referring to the files selected by the user.

Example

```
choose several files ¬
    with prompt "Select files to be archived:" ¬
    of type {"APPL", "TEXT" } ¬
    starting with { alias "Hard Disk:Disinfectant" }
```

Choose Several Folders

The Choose Several Folders command presents a modified standard file selection dialog box allowing the user to choose several folders at one time.



Syntax

```
choose several folders
    [ with prompt promptString ]
    [ starting with folderList ]
```

Parameters

promptString This parameter is a string which is displayed in the dialog box. If you omit the `with prompt` parameter, no prompt is displayed.

folderList This parameter is a list of aliases referring to folders which are to be displayed as already selected. If you omit the `starting with` parameter, the selected folders list is left empty.

Result

The result is a list of aliases referring to the folders selected by the user.

Example

```
choose several folder ¬
    with prompt "Select files to be archived:" ¬
    starting with ¬
        { alias "HD:System Folder:" ¬
          alias "HD:System Folder:Extensions:" }
```

Get Default Folder

The Get Default Folder command returns the current folder used by the Choose File and Choose Folder commands in this package and those provided by Apple as part of AppleScript.

Syntax

```
get current folder
```

Result

This command returns an alias to the current default folder.

Example

```
set saveFolder to get default folder
set default folder path to preferences
choose file
set default folder saveFolder
```

Set Default Folder

The Set Default Folder command changes the current folder used by the Choose File and Choose Folder commands in this package and those provided by Apple as part of AppleScript.

Syntax

`set default folder` *folderPath*

Result

This command returns no result.

Parameters

folderPath This parameter is an alias to the folder which is to become the default folder. If you provide an alias to a file, the folder containing the file becomes the default folder.

Example

```
set default folder path to preferences
choose file
```

Shutdown

The Shutdown command shuts down and optionally restarts your Macintosh.

Syntax

```
shutdown
    [ with restart ]
```

Result

This command returns no result.

Example

```
set result to display dialog ¬
    "Are you sure you want to shutdown?" ¬
    buttons {"Shutdown", "Restart", "Cancel"} ¬   default
button "Cancel"
if button returned of result = "Shutdown" then ¬ shutdown
if button returned of result = "Restart" then ¬  shutdown
with restart
```

Open File

The Open File command opens a text file for reading and/or writing. This command, when used with the Read File and Write File commands, allows you to process text files within scripts without the aid of a scriptable text editor application.

Syntax

```
open file file
    [ for reading|update|writing ]
```

Parameters

file This parameter is a alias to the file which is to opened.

Result

The result a file reference number. You must provide this number to all other commands you issue when processing the file.

Example

```
set filePath to choose file ¬
   with prompt "Select a file to open:" ¬
   of type "TEXT"
set refNum to open file filePath for reading
close file refNum
```

Notes

When the optional `for` is not specified, the file is opened for update.

Be careful to ensure you close all the files you open. Due to the nature of AppleScript additions, the Open File command does not ensure the file is closed when a script aborts without first closing the file with the Close File command.

Errors

This command can return any of the errors which are returned by the ToolBox HOpen routine.

Close File

The Close File command closes a file previously opened with the Open File command.

Syntax

`close file` *fileRefNum*

Parameters

fileRefNum This parameter is the reference number of a file. This value is returned by the Open File command.

Result

none

Example

```
set filePath to choose file ¬
   with prompt "Select a file to open:" ¬
   of type "TEXT"
set refNum to open file filePath for reading
close file refNum
```

Errors

This command can return any of the errors which are returned by the ToolBox FSClose routine.

Create File

The Create File command creates a new TEXT file.

Syntax

```
create file fileName
    [ in folder ]
    [ owner signature ]
```

Parameters

fileName	This parameter is the new file's name.
folder	This parameter is an alias to the folder where the new file is to be placed. If this parameter is omitted the file is created in the current default folder.
signature	This parameter is a list of aliases referring to folders which are to be displayed as already selected. If you omit the `owner` parameter, the new file is given the signature `'????'`.

Result

none.

Example

```
set newFile to choose new file ¬
    with prompt "Pick a new file name:"

create file (filename returned of newFile) ¬
    in (folder returned of newFile) ¬
    owner "ttxt" -- TeachText
```

Errors

This command can return any of the errors which are returned by the ToolBox HCreate routine.

Create Folder

The Create Folder command creates a new folder.

Syntax

```
create folder folderName
    [ in folder ]
```

Parameters

folderName This parameter is the new folder's name.

folder This parameter is an alias to a folder where the new folder is to be placed. If this parameter is omitted the file is created in the current default folder.

Result

none.

Example

```
set newFolder to choose new file ¬
    with prompt "Pick a new folder name:"

create folder (filename returned of newFolder) ¬
    in (folder returned of newFolder)
```

Errors

This command can return any of the errors which are returned by the ToolBox DirCreate routine.

Delete File

The Delete File command deletes a file without placing it in the Trash.

Syntax

```
delete file folders
    [ with prompt promptString ]
    [ starting with folderList ]
```

Parameters

promptString This parameter is a string which is displayed in the dialog box. If you omit the `with prompt` parameter, no prompt is displayed.

folderList This parameter is a list of aliases referring to folders which are to be displayed as already selected. If you omit the `starting with` parameter, the selected folders list is left empty.

Result

The result is a list of aliases referring to the folders selected by the user.

Example

```
choose several folder ¬
    with prompt "Select files to be archived:" ¬
    starting with ¬
        { alias "HD:System Folder:" ¬
          alias "HD:System Folder:Extensions:" }
```

Errors

This command can return any of the errors which are returned by the ToolBox HDelete routine.

Rename File

The Rename File command changes a files name.

Syntax

`rename file` *file* `to` *newName*

Parameters

file	This parameter is a alias which identifies the file whose name is being changed.
newName	This parameter is a text string containing the file's new name.

Result

none.

Example

```
rename file ( choose file ) to "Backup"
```

Errors

This command can return any of the errors which are returned by the ToolBox PBHRename routine.

Exchange File

The Exchange File command swaps the data stored in two files.

Syntax

`exchange file` *firstFile* `with` *secondFile*

Parameters

firstFile This parameter is a alias which identifies the first of the two files.

secondFile This parameter is a alias which identifies the second of the two files.

Result

none.

Example

```
exchange file (choose file) with (choose file)
```

Errors

This command can return any of the errors which are returned by the ToolBox PBExchangeFiles routine.

Move File

The Move File command moves a file or a folder from one folder to another.

Syntax

`rename file` *fileOrFolder* `to` *destination*

Parameters

fileOrFolder This parameter is an alias which identifies the file or folder being moved.

destination This parameter is an alias referring to the destination folder for *fileOrFolder*.

Result

none.

Note

The file or folder being moved and the destination folder must be on the same volume.

Example

```
move file ( choose file ) to ( choose folder )
```

Errors

This command can return any of the errors which are returned by the ToolBox PBCatMove routine.

Read File

The Read File command reads a “line” of text from a file opened with the Open File command. A line in this case means all characters up to the next carriage return in the file. This is refereed to as a paragraph in some applications since these lines may wrap around a number of times when displayed in a window.

Syntax

```
read file fileRefNum
    [ maximum length maxLength ]
```

Parameters

fileRefNum This parameter is the reference number of a file. This value is returned by the Open File command.

maxLength This integer parameter specifies the maximum number of characters you wish to read. Normally the Read File command reads a maximum of 1024 characters. The practical maximum for this value is limited only by the memory available.

Result

The result is a string representing the data read from the file.

Example

```
set myFile to choose file ¬
    with prompt "Select a text file:" ¬
    of type "TEXT"
set refNum to open file myFile
set inputLine to read file refNum
display dialog inputLine
close file refNum
```

Errors

This command can return any of the errors which are returned by the ToolBox PBRead routine.

Write File

The Write File command writes a line to a text file.

Syntax

`write file` *fileRefNum* `text` *data*

Parameters

fileRefNum	This parameter is the reference number of a file. This value is returned by the Open File command.
data	This parameter is the line of text to be written to the file.

Result

none.

Example

```
set refNum to open file "Sample Test"
write file refNum text "Sample Test"
close file refNum
```

Errors

This command can return any of the errors which are returned by the ToolBox FSWrite routine.

Get File Position

The Get File Position command obtains the current position of a file's marker. A file marker represents the address within a file where the next read or write will begin.

Syntax

`get file position` *fileRefNum*

Parameters

fileRefNum This parameter is the reference number of a file. This value is returned by the Open File command.

Result

The result is a number representing the address of the files marker.

Example

```
-- haven't thought of a good one yet
```

Errors

This command can return any of the errors which are returned by the ToolBox GetFPos routine.

Position File

The Position File command changes the current position of a file's marker. A file marker represents the address within a file where the next read or write will begin.

Syntax

`position file` *fileRefNum* `at` *filePosition*

Parameters

fileRefNum This parameter is the reference number of a file. This value is returned by the Open File command.

filePosition This parameter is the new address for the files marker.

Result

none.

Example

```
-- position the marker at the end of the file so
-- data can be appended to the file
position file refNum to (get length refNum)
```

Errors

This command can return any of the errors which are returned by the ToolBox SetFPos routine.

Get Length

The Get Length command obtains the length (in bytes) of the file.

Syntax

```
get length fileRefNum
```

Parameters

fileRefNum This parameter is the reference number of a file. This value is returned by the Open File command.

Result

The number of bytes stored in the file.

Example

```
-- position the marker at the end of the file so
-- data can be appended to the file
position file refNum to (get length refNum)
```

Errors

This command can return any of the errors which are returned by the ToolBox GetEOF routine.

Lengthen File

The Lengthen File command changes the length of a file. You can use the Lengthen command to shorten or extend the size of a file.

Syntax

```
lengthen file fileRefNum length fileLength
```

Parameters

fileRefNum This parameter is the reference number of a file. This value is returned by the Open File command.

fileLength This parameter is the new length of the file.

Result

none.

Example

```
-- empty the contents of a file
lengthen file refNum length 0
```

Compile Regular Expression

The Compile Regular Expression command compiles a pattern string. Compiled Regular Expressions are used by the Match Regular Expression and Substitute Regular Expression commands.

Syntax

`compile regular expression` *patternString*

Parameters

patternString This parameter is a string which is displayed in the dialog box. If you omit the `with prompt` parameter, no prompt is displayed.

For a description of the syntax of pattern strings see the documentation for the UNIX grep command. Information about Regular Expressions is also available in the THINK C User's Guide.

Result

The result is a compiled version of the patternString. This compiled pattern is used with the Match Regular Expression and Substitute Regular Expression commands.

Example

```
set pattern to
   compile regular expression "(.*):(*)"
```

Errors

V1.2 of Compile Regular Expression does not report any errors. If there is a problem with the pattern string a null expression ("") is returned. Future releases will return errors indicating the type of problem found with the pattern string.

Match Regular Expression

The Match Regular Expression command matches a string to a Regular Expression and returns the portions of the string which match the regular expression.

Syntax

```
match regular expression compiledExpression
    to candidateString
```

Parameters

compiledExpression

This parameter is a compiled regular expression. This value is returned by the Compiler Regular Expression command.

candidateString

This parameter is the string that is to be matched to the regular expression.

Result

The result of the Match Regular Expression command is a record containing the following values:

`matched`

This Boolean value indicates if there was a match.

`match string`

This string value represents largest match found.

`match 1` This string value represents the portion of the string matching the first () expression.

`match 2` This string value represents the portion of the string matching the second () expression.

`match 3` This string value represents the portion of the string matching the third () expression.

`match 4` This string value represents the portion of the string matching the fourth () expression.

`match 5` This string value represents the portion of the string matching the fifth () expression.

`match 6` This string value represents the portion of the string matching the sixth () expression.

`match 7` This string value represents the portion of the string matching the seventh () expression.

`match 8`	This string value represents the portion of the string matching the eighth () expression.
`match 9`	This string value represents the portion of the string matching the ninth () expression.

Example

```
set pattern to ¬
    compile regular expression "This (.*) test"
set result to match regular expression pattern ¬
    to "This is a test"
{ result }
```

Output formatted for this document:

```
{
    matched : TRUE,
    matched string: "This is a test",
    match 1: "is a"
}
```

Substitute Regular Expression

The Substitute Regular Expression command extracts the elements from a candidate string which match the patterns of a Regular Expression and then substitutes the extracted elements into a template string.

Syntax

`substitute regular expression` *compiledExpression*
    `of` *candidateString*
    `with` *templateString*

Parameters

compiledExpression
: This parameter is a compiled Regular Expression pattern. Regular Expressions are compiled using the Compile Regular Expression command.

candidateString
: This parameter is a string representing the text which is to be compared to the Regular Expression and then modified.

templateString
: This parameter is a string representing a template for the substitutions which are to be performed. See the section titled "Replacements for Regular Expressions" below for a description of the format of this string.

Result

The result is the substituted string.

Example

```
set pattern to ¬
    compile regular expression "This (.*) test"
substitute regular expression pattern ¬
    of "This is a test" with "---\\1---"
```

Result:

```
---is a---
```

Replacements for Regular Expressions

Within a template string the following conventions apply:

- A backslash quotes the following character. The special characters within a template string are '`&`' and '`\`'; these are the only characters that need to be quoted. The construct "`\&`" produces a single '`&`' and the construct "`\\`" produces a single backslash.
- An ampersand (&) indicates the entire matched regular expression. For example, the replacement "`&&`" would consist of two copies of the matched expression.
- The sequence "`\`*`n`*", where n is a single digit, indicates the text matching the *n*th parenthesized component of the regular expression

Speak

The Speak command uses the Apple Macintosh Speech Manager to speak text strings. Note that because of its dependency on the Speech Manager, this command only operates on Macintoshes which have the Speech Manager installed.

Syntax

```
speak message
    [ voice voice ]
    [ rate rate ]
    [ pitch pitch ]
```

Parameters

message This parameter is the text you want to have spoken.

voice	This optional parameter allows you to specify the name of the voice you want used when the message is spoken.
rate	This optional parameter specifies the rate at which your message is spoken. Express the rate as a number representing words per minute.
pitch	This optional parameter specifies the pitch at which your message is spoken.

Result

none.

Example

```
speak "The wind blows mainly in the plains"
```

List Voices

The List Voices command obtains a list of the names of the voices available. Note that because of its dependency on the Speech Manager, this command only operates on Macintoshes which have the Speech Manager installed.

Syntax

```
list voices
```

Parameters

none.

Result

The result is a list of strings representing the names of all the Speech Manager voices.

Example

```
list voices
```

Result:

```
{"Mr. Hughes", "Xero", "Votron", "Otis", "RoboVox",
"Boris", "Mariel", "Ben", "Brenda", "Marvin"}
```

Get Voice

The Get Voice command returns detailed information about a particular Speech Manager voice. Note that because of its dependency on the Speech Manager, this command only operates on Macintoshes which have the Speech Manager installed.

Syntax

```
get voice voice
```

Parameters

voice This parameter specifies the name of the voice you want information about.

Result

The result of the Get Voice command is a record containing the following values:

`voice version`
: This integer value represents the voice's version number.

`voice name`
: This string is the voice's name.

`comment` This string further describes the voice.

`gender` This integer value defines the gender of the voice—1 = neuter, 2 = male and 3 = female.

`age` This integer value represents the approximate age of the voice.

`voice script`
: This integer corresponds the voice's script code.

`language` This integer value is the voice's language code.

Example

```
get voice (first item of (list voices))
```

Output formatted for this document:

```
{
    version:65536,
    name:"Mr. Hughes",
    comment:"Adult male voice.",
    gender:1,
    age:30,
    script:0,
    language:0,
    region:0
}
```

Get Gestalt

The Get Gestalt command gets information about the operating environment.

Syntax

```
get gestalt selector ]
    [ bit bitNumber ]
    [ with/without report missing selectors ]
```

Parameters

selector This parameter is a string representing the type of operating environment information you want. This parameter must be a 4-character code. The gestalt Selectors Lib file defines all of the Gestalt selectors documented in Inside Macintosh volume VI as well as selectors for Apple's Speech Manager.

bitNumber This optional parameter defines which bit of the selectors value to test. If this parameter is specified the command returns a Boolean value. If the parameter is omitted the command returns the entire selector value.

Result

The result of this command is either the selector's integer value when the `bit` parameter is not specified. When the `bit` parameter is specified a Boolean value is returned.

Notes

When the optional `with report missing selectors` is specified, the Get Gestalt command reports errors associated with unknown selectors Otherwise a value of 0 is returned.

Example

```
-- verify that the Speech Mgr is present

property gestaltSpeechAttr : "ttsc"
property gestaltSpeechMgrPresent : 0

if get gestalt gestaltSpeechAttr ¬
   bit gestaltSpeechMgrPresent then
    display dialog "Speech Mgr Present"
else
    display dialog "Speech Mgr Missing"
end if
```

List Processes

The List Processes command obtains a list of the names of the applications running on your Macintosh. This includes normal Macintosh applications, desk accessories and faceless-background-only applications.

Syntax

```
list processes
```

Parameters

none.

Result

The result is a list of strings representing the names of all the running applications.

Example

```
list processes
```

Result:

```
{"PrintMonitor", "Scheduler", "File Sharing Extension",
"Finder", "Eyes", "Monitor", "Script Editor"}
```

Get Process

The Get Process command obtains detailed information about a running application.

Syntax

`get process` *processName*

Parameters

processName This parameter specifies the name of the process you want information about.

Result

The result of the Get Process command is a record containing the following values:

`process name`
: This string is the process's name.

`process number`
: This value represents the process's serial number. AppleScript translates this value into an application object automatically.

`application type`
: This string is the application's four character file type. Normally this value is "APPL".

`signature`
: This string is the application's four character signature.

`partition size`
: This integer value represents the amount of memory the application occupies.

`free memory`
: This integer value represents the amount of free memory within the application's partition.

`launcher`
: This string is the name of the application which launched this application. If its blank then the application is no longer running.

`launch date`
: This integer value represents the data and time when the application was launched.

`active time`
: This integer value is the amount of CPU time used by the application since it was launched. The units for this value are ticks (1/60th of a second).

`application file`
: This value is a reference to the application's file.

`deskAccessory`

`multiLaunch`

`needSuspendResume`

`canBackground`

`activateOnForegroundSwitch`

`compatible32Bit`

`onlyBackground`

`getFrontClicks`

`getApplicationDiedEvents`

`highLevelEventAware`

`localAndRemoteEvents`

`stationeryAware`

`useTextEditServices`
: These Boolean values represent the application's mode flags.

Example

```
get process (first item of (list processes))
```

Output formatted for this document:

```
{
    process name:"PrintLauncher",
    process number:application "PrintLauncher",
    application type:"appe",
    signature:"PRLN",
    partition size:40960,
    free memory:5018,
    launch date:date "Saturday, September 11, 1993
10:23:45 AM",
    active time:19605,
    application file:file "System Disk:System
Folder:Extensions:PrintLauncher",
    deskDccessory:false,
    multiLaunch:false,
    needSuspendResume:true,
    canBackground:true,
    activateOnForegroundSwitch:true,
    onlyBackground:true,
    getFrontClicks:false,
    getApplicationDiedEvents:false,
    compatible32Bit:true,
    highLevelEventAware:true,
    localAndRemoteEvents:false,
    stationeryAware:false,
    useTextEditServices:false,
    launcher:""
}
```

Get Foreground Process

The Get Foreground Process command gets name of the foreground application. The foreground application is the application whose windows are presently active. Note that the foreground application is not necessarily the current application (see the Get Current Application command).

Syntax

```
get foreground process
```

Parameters

None.

Result

The result of this command is string representing the name of the foreground application.

Example

```
get foreground process
```

Result:

```
"Script Editor"
```

Get Current Process

The Get Current Process command gets name of the currently executing application. This command is useful for finding the name of the process executing a script. The value returned by the Get Current Process command is different from the value returned by the Get Foreground Process command when the current process is in the background.

Syntax

```
get current process
```

Parameters

None.

Result

The result of this command is string representing the name of the currently executing application.

Example

```
get current process
```

Result:

```
"Script Editor"
```

List Screens

The List Screens command obtains detailed information about each of the Macintosh's display screens.

Syntax

```
list screens
```

Result

The result of the List Screens command is a list of records. Each record describes a different display screen. The records contain the following values:

`main screen` This Boolean value indicates weather or not the screen is the main screen. The main screen is the screen containing the menu bar.

`bit depth` This value represents the number of bits in the display screen.

`bounds` This value is the screens bounding rectangle.

Example

```
list screens
```

Output formatted for this document:

```
{
    {
        main screen:true,
        bit depth:1
        bounds:{0, 0, 1152, 882}
    },
    {
        main screen:false,
        bit depth:1
        bounds:{-512, 356, 0, 698}
    }
}
```

Addition/Command Cross Reference

The following table lists the commands defined in each of the ScriptTools additions:

Choose Files & Folders	Choose Folder Choose New File Choose Several Folders Choose Several Files
File IO	Open File Close File Create File Create Folder Delete File Rename File Exchange File Move File Read File Write File Get File Position Position File Lengthen File Get Length
Shutdown	Showdown
Regular Expressions	Compile Regular Expression Match Regular Expression Substitute Regular Expression
Speech	Speak List Voices Get Voice
Gestalt	Get Gestalt
Processes	List Processes Get Process Get Current Process Get Foreground Process
List Screens	List Screens