GTQ Scripting Library

Version 1.1
1 April 1994
Gregory T. Quinn

INTRODUCTION

Welcome to version 1.1 of the GTQ Scripting Library for Apple Computer's AppleScript software. I hope you find some of these scripting extensions (osax) useful.

The following extensions perform tasks which the new scriptable Finder can do:

Application Info for	Make Alias	Version of (similar)
Relocate	Remove	Current Date in Seconds (obsolete)
Rename	List Applications (similar)	

The two marked similar provide slightly different functionality than the scriptable Finder. List Applications lists all running processes on the local machine while the Finder can list the processes displayed in the About This Macintosh window. Version of extracts either version (long or short) while the scriptable Finder can only extract the long version but also can return other information which appears in the Get Info window. The current date in seconds command is unnecessary with AppleScript 1.1 and version 1.1 of my date string for and time string for extensions. If you think I have missed any overlaps or misrepresented the scriptable Finder's (or my library's) functionality let me know!

Some people prefer scripting extensions which are grouped together into one file; therefore, I now provide the GTQInstaller (a Frontmost standalone) which supports the packaging of *my* osaxen into single files. See **Installation** below.

You may distribute my library in any way in the same exact form I originally posted it in (documentation, all extensions, and all sample scripts). You may only distribute it for non-commercial purposes. You are not allowed to profit from the distribution of my library. This library is free for anyone who wants it. Anyone who insists on licensing/compensation must contact me. No additional support is given to people who compensate me. Mere contact usually earns additional support. If you wish to include some portion of my library in something you will be selling, please contact me. If you have a sample script that better demonstrates the use of some of my osaxen (not so difficult, I know), I would love to include it (with your name on it of course).

Please feel free to contact me for questions or comments. I appreciate all bug reports, enhancement suggestions, and new extension suggestions.

Greg Quinn
AppleLink: D3297
Internet: gtq1@cornell.edu

I want to thank Sean McMains and Chris Ogden for their support and encouragement. I also want to thank

GTQ Scripting Library Version 1.1

Apple Computer for making the best operating system even better with OSA and AOCE.

INSTALLATION

Drag the scripting extensions you want to use directly into the Scripting Additions folder which resides within the Extensions folder (:System Folder:Extensions:Scripting Additions:).

If you want to package multiple osaxen from my library into a single file, double click on the GTQInstaller, click ok to dismiss the dialog box which appears, then choose "Package…" from the "File" menu. Select the folder containing the individual osaxen and the Frontmost application will combine them into one file in the Scripting Additions folder within the Extensions folder in the System Folder. The original osaxen will not be deleted or affected in any way by this process. The packaging is not reversible currently.

SAMPLE SCRIPTS

All of the sample scripts are compiled for AppleScript 1.0 because AppleScript 1.1 scripts (compiled) will not run on machines with AppleScript 1.0 installed. However, significant performance gains can result from re-compiling some of the scripts for AppleScript 1.1. In particular, the Server Monitor Daemon 1.2 (SMD 1.2) script by Sean McMains will execute faster if re-compiled for AppleScript 1.1. I attribute most of the performance gains seen to the new data structure used internally for lists in AppleScript 1.1 (vectors versus linked lists). Nice job AppleScript team!

SUMMARY OF CHANGES

Version 1.0 to 1.1:

- library grew from 41 to 57 osaxen. new: add picture, add resource, add string list, append aete, audio CD, does font exist, extract number from, extract picture, extract resource, extract string list, font information for, format number, list fonts, list resource types in, number of resources in, and object database
- address of and choose address both version 1.1.0 and mail to version 1.2.0.
- sort version 1.0.1
- choose from list version 1.0.1
- choose link version 1.0.1
- set printer to version 1.0.1
- date string for and time string for now support Apple's current date osaxen: both are version 1.1.0
- reorganization of this manual: grouped osaxen and made manual a Word outline.

For a complete history of the GTQ Scripting Library, send me a message and I will forward you my revision history.

Application Osaxen

Application Info for 20 1.0.0r2

Input: application info for <application specification>

Output: <record> containing {name:<string>, type:<string>, signature:<string>,

mode:<integer>, size:<integer>, time:<integer>, file:<file specification>}

This command simply returns the above record for the specified *local running* application. The application can be specified by a standard AppleScript application specification:

```
set appInfo to (application info for application "Finder")
```

In addition, because of the **String To Application Coercion Handler** (the other way around is built into AppleScript), the following is equivalent:

```
set appInfo to (application info for "Finder")
```

The name field in the record is a string containing the application's name. The type field is the file type of the application (not always "APPL"). The signature field is the application's unique creator code; e.g., the Finder's signature is "MACS". The mode value gives information about the application that the operating system uses (the "SIZE" resource flags). Most users can ignore this result. The size parameter indicates the amount of memory the application has allocated to it. The time value equals the number of seconds the application has been running. The file spec field contains the file specification for the application's file on disk.

Front Application **29** **1.0.0r1**

Input: front application
Output: <application specification>

To determine the application the user is currently providing input to, i.e., the front most application or the application whose menus are displayed, execute this command. Compare the result of this command to your script application's specification to see if your script is in the background or foreground. **This Application** returns the script's application specification. If you are in the background and need to interact with the user, use the **Request Attention** command described below. The following example is useful for providing user interaction in a script which may be running in the background:

```
if (front application) ≠ (this application) then request attention message "I need your input!"
```

Is Application Running **7** **1.0.0r2**

Input: is application running <string>
Output: <boolean>

This command returns true if the application specified by the input string is currently running on the local machine. The string comparison to determine if the string is the name of a running application is case insensitive in versions 1.0r2 or later. Scripts containing this command are not very portable because application names can vary from machine to machine.

List Applications **19** **1.0.0r1**

Input: list applications
Output: <list> containing application specifications

This command outputs a list of applications running on the local machine. The list contains application specifications, e.g.:

```
application "Finder"
```

Any item in this list can then be used in a tell statement or in the **Application Info for** command to name some examples.

String to Application Coercion Handler **22** **1.0.0r1**

This coercion handler allows you to use the name of an application in place of an application specification anywhere one is required as input. The case of the string is ignored.

Switch to Launcher **6** **1.0.0r1**

Input: switch to launcher
Output: none

This command simply brings the application (usually the Finder) which launched the current application to the front.

This Application **30** **1.0.0r1**

Input: this application
Output: <application specification>

To determine the application the script is running within. If the script is an applet, the applet's application specification will be returned. See **Front Application** for an important use of this command.

Version of **25** **1.0.0r2**

Input: version of <file specification>[in <format enumeration>]
Output: <list> containing version numbers found (max is 2)
<format enumeration>:
long format/short format

This command attempts to extract version strings from the specified file. If two versions exist, it gets both of them if they have resource IDs of 1 and 2 (the Apple-specified numbering scheme). The first will always correspond to the actual file's version while the second is the family's version. For example, this library has versions for each individual command, but the entire library also has a version number corresponding to the latest release. While the first version number will differ from addition to addition, the second number should always be the same for all of my extensions. If they aren't, you don't have the latest version of one of the commands.

The format enumeration allows extraction of the long version strings or short version strings. The file's long version string appears next to the title "Version:" in the Get Info window for the file. The family's long version string appears on the second line of the Get Info window beneath the application's name. A file's short version string appears in the version column of the Finder's list views. The family's short version string doesn't appear anywhere that I know of.
Date/Time Osaxen

Current Date in Seconds 28 1.0.0r1

Input: current date in seconds
Output: <real>

This command returns the number of seconds since midnight, January 1, 1904. The result is placed in a real variable so that durations can be calculated:

```
set startTime to (current date in seconds)
…
set duration to (current date in seconds)-startTime
```

In addition, the values returned by this command can be passed to the **Date String for** and **Time String for** commands to get a string representing the date or time respectively. The fractional part of the result is not meaningful.

Date String for 38 1.1.0r2

Input: date string for <real>/<date>[in <format enumeration>]
Output: <string>
<format enumeration>:
short format/long format/abbreviated format

This command returns a string representing the date specified by the number of seconds passed in the first parameter. The date is formatted according to the current system software in use; i.e., the international utilities package performs the formatting for me. The system software allows three separate date formats: short, long, or abbreviated. If the format parameter is left out, the short format is used.

The formats appear as follows on my system using standard U.S. system software (Roman script). I should note that these formats *only* apply to the default formats. These formats can be customized via the Date & Time control panel and will therefore be different.

short format

The short format on my system appears as "mm/dd/yy" where mm specifies the number of the month, dd the number of the day within that month, and yy is the last two digits of the year.

```
date string for (current date in seconds) in short format
    => "1/22/94"
```

long format

The long format on my system appears as "wwww, mmmm dd, yyyy" where wwww specifies the name of the day of week fully written out, mmmm specifies the name of the month fully written out, dd specifies the day number in that month, and yyyy specifies the year with all four digits.

```
date string for (current date in seconds) in long format
    => "Saturday, January 22, 1994"
```

abbreviated format

The abbreviated format on my system appears as "www, mmm dd, yyyy" where www specifies the first three letters for the day of the week, mmm specifies the first three letters for the month, dd specifies the day number in that month, and yyyy specifies the year with all four digits.

```
date string for (current date in seconds) in abbreviated format
    => "Sat, Jan 22, 1994"
```

With version 1.1.0 of this extension, the command can now except the date value class which is supported internally by AppleScript and returned by Apple's current date osax.

Time String for **39** **1.1.0r2**

Input: time string for <real>/<date>[with/without seconds]
Output: <string>

This command returns a string representing the time specified by the number of seconds passed in the first parameter. The time is formatted according to the current system software in use; i.e., the international utilities package performs the formatting for me. The format can be customized via the Date & Time control panel. The parameter specifies whether to format the time with the seconds or without the seconds. If the parameter is left out, it defaults to without seconds.

time string for (current date in seconds) without seconds
=> "12:50 PM"

time string for (current date in seconds) with seconds
=> "12:49:46 PM"

With version 1.1.0 of this extension, the command can now except the date value class which is supported internally by AppleScript and returned by Apple's current date osax.
Files Osaxen

Make Alias **2** **1.0.0r2**

Input: make alias <file specification>[named <string>][at <file specification>]
Output: <alias>

To create a Finder alias file, use this command. The first parameter is a specifier for the file to make the alias of. The optional name parameter is a name for the new alias file. This parameter defaults to the original file's name with " alias" appended to it as the Finder always uses. The final parameter allows specification of a different location for the new alias file. If this parameter is left out, the new alias file appears in the same directory as the file it points to (the one specified by the first parameter). If a file of the alias file's name already exists in the alias file's destination, an error results. A future version will allow options to handle this case better (unless the scriptable Finder appears soon). The routine also returns an alias specification to the original file.

Relocate **3** **1.0.0r1**

Input: relocate <file specification> to <file specification>
Output: none

This command moves the file indicated in the first parameter to the location, which must be on the same volume, specified by the second parameter. If a file by that name already exists at that location, an error results currently. A future version will allow options to handle this case better (unless the scriptable Finder appears soon).

Remove **27** **1.0.0r1**

Input: remove <file specification>[with <verification enumeration>]
Output: <boolean>
<verification enumeration>:
verification/no verification

This command deletes a file or a folder. The first parameter specifies which file or folder to delete. The second parameter dictates whether the file deletion request is verified first or not. The default for this optional parameter is verification; i.e., if the second parameter is left out (or "with verification" is used), the deletion will be verified by a caution alert before being performed. The function returns a true value if the file/folder was actually deleted and returns false otherwise. The first example below would bring up a caution alert before deleting and the second one would not:

```
remove file "HD:Desktop Folder:DeleteME" with verification
remove file "HD:Desktop Folder:DeleteME" with no verification
```

Rename **5** **1.0.0r1**

Input: rename <file specification> to <string>
Output: none

This addition renames a file given by the first parameter to the string in the second parameter. If a file of the same name as the string already exists in the same directory as the file, an error results. A future version will allow options to handle this case better (unless the scriptable Finder appears soon).
Fonts Osaxen

Does Font Exist **53** **1.0.0r1**

Input: does font exist <string>[of size <small integer>]
Output: <boolean>

This command checks for a specified font. The first parameter is the font's name. The name is case insensitive (i.e., Courier and courier are equivalent). If the second parameter is not included, the command checks to see if the font of the specified name is available on the current system and returns true if it is and false otherwise.

If the optional second size parameter is included, the command checks to see if the specified font in the indicated size is available. For a font of a particular size to be available, a bitmap must exist in that size or a TrueType version of the font must exist. In addition, if the font size is small and no bitmap exists for it, the requested size must be larger the smallest readable size for the specified font. For example, even TrueType fonts can scale a font down to a point size of 2 (in general). The following script determines the smallest valid font size for a chosen font:

```
set theFont to (choose from list (list fonts))
```

```
set i to 0
set res to false
repeat while not res
	set i to i + 1
	set res to (does font exist theFont of size i)
end repeat

display dialog "That smallest size you can use " & theFont & " in is " & i & "."
```

Font Information for 52 1.1.0r3

Input: font information for <file specification>[returning <format enumeration>]
Output: <list> of <record> containing {family name:<string>, family ID:<small integer>, font list:<list> of <record> containing {style:<string>,[point size:<small integer>], resource id:<small integer>, font type:<string>, PostScript font name:<string>}}
<format enumeration>:
everything/font families/font families and styles

This command returns information about the fonts contained in the file (suitcase or font file) specified by the first parameter. If the second parameter is "font families," only a list of the font families which exist in the file are returned. If this second parameter is "font families and styles," a list of records is returned. Each record represents on font family in the file and contains the name of the family in addition to a list of the styles for the family available in the file. A list of records describing each font family in the file is returned if the second parameter is left out or if "everything" is passed to it. Each record describing a family contains the following fields:

family name	Font family's name
family id	Font family's resource id
style list	List of descriptions of members of the family in this file.

The final field consists of a list of style records describing the available styles for the family. These records take the following format:

style	Font's style (e.g., "Plain", "Bold", or "Bold, Italic"). A comma always separates styles and each has the first letter capitalized. These style strings appear in a "STR#" resource in the osax's file.
PostScript font name	Font's PostScript name equivalent.
font list	Records describing the actual fonts available for this style in this family

The final field of this record also contains a list. This list provides information on the actual bitmapped and outline (TrueType) fonts available in the style for the family. This record has the following format:

resource id	Resource id for the font's 'sfnt', 'NFNT', or 'FONT' resource.
font type	The resource type for the font: 'sfnt', 'NFNT', or 'FONT'. If the type is 'sfnt', the font is an outline (TrueType) font; otherwise, it is a bitmapped font.
point size	Font's point size or zero if the font is an outline font (the above field will be 'sfnt')

List Fonts 51 1.0.0r1

Input: list fonts

Output: <list> containing strings

This command outputs a list of fonts available on the local machine. Any fonts which would appear in an application's font menu will appear in this list.
Formatting Osaxen

Extract Number from **55** **1.0.0r1**

Input: extract number from <string> in format <string>
Output: <extended>

This command allows you to "parse" a specific string for a number contained in it. The first parameter is the string containing the number and the second parameter is a string specifying the format the first string is in. The command returns the number.

The format string takes an identical format to the format string described in the **Format Number** command. Here is an example of using this command:

```
extract number from "Total=($500.23)" in format "'Total=$'###.##;'Total='('$'###.##')'"
    => -500.23
```

Format Number **54** **1.0.0r1**

Input: format number <extended> in format <string>
Output: <string>

This command provides international number formatting. A number is passed in the first parameter along with a format specification string and a string containing the number in the desired format is output.

The format specification string defines the appearance of the numeric string. These strings are also described in *Inside Macintosh - Text* on pages 5-39 through 5-43. The string can contain from one to three parts where parts are separated by semicolons. If one part is specified, this format string is used for all inputs. If two parts are specified, the first one is used for positive numbers and zero, and the second part is used for negative numbers. If three parts are given, the first one is used for positive numbers, the second for negative numbers, and the third for zero. When no specific negative format specification part is given, a negative sign is used in front of the positive format specification part. The following items appear within a part's format specification:

- the decimal separator and the thousand separator as defined in the Numbers control panel
- literals which are enclosed in single quotes (plus escape sequence for including quotes in literals)
- special literals which do not require quotes: (),[], and {}
- digit placeholders: #, 0, or ^ (see below)
- symbol and sign characters

Separators include the decimal point or the thousand separator. The separator you use must match the separators defined in the Numbers control panel.

Literals are quoted strings or brackets, braces, or parentheses. Because quoted literals are enclosed in single

quotes, an escape sequence is necessary to produce single quotes within strings. The backslash is the escape character; i.e., backslash then a single quote contained within a quoted literal will produce a single quote in the output string (no backslash will appear).

The three digit placeholders not only indicate where digits appear but also how many digits can appear. For integer digits (ones to the left of the decimal place), if more digits are required than you specify placeholders for, an error will result. If more digits are needed for decimal digits, the number is rounded to fit. The three digit placeholders are zero digits (0), skipping digits (#), and padding digits (^):

- zero digits force leading zeros when no digit is present in its place.
- skipping digits will be skipped unless a digit actually exists for the place but specify a digit is allowed in the place
- padding digits work like zero digits except instead of a zero, a padding character is used in the place as defined by the current script system. A common padding character is the nonbreaking space.

Integer digits are filled in from right to left and decimal digits from left to right.

Symbol and sign characters depend on the script system, but they can also be used within format specification strings. The symbols defined for the Roman script system are: +, -, %, E+, and E-. The last two force formatting of the numbers using scientific notation.

Some examples would be very helpful here:

```
format number 5.647 in format "##.##;(##.##)"
    => "5.65"

format number -325 in format "###.00"
    => "-325.00"

format number -4000 in format "' '^^#.##' ';(^^#.##)"
    => error "The number in the input string exceeded the magnitude allowed in the number format specification."

format number -42.5 in format "' '^^^.00' ';(^^^.00);'ZERO!'
    => "( 42.50)"
```

Miscellaneous Osaxen

Choose from List **41** **1.0.1r3**

Input: choose from list <list>[prompt <string>][multiple selections <boolean>][ok button <string>][cancel button <string>][empty lists <boolean>][initially select <list>]

Output: <list> containing the chosen items or false

This addition presents a dialog box containing a user-selectable list of the items contained in the first parameter's list. They appear in the same order as they appear in the list. The prompt displayed can be passed in the second parameter or a default prompt will be used. The third parameter is also optional and specifies whether the user can select more than one item from the list. If multiple selections are allowed, the shift key extends selections in blocks and the command key adds specific items possibly forming disjoint selections. The available selection methods are standard Macintosh.

The next two parameters are also optional and specify the actual titles of the two buttons: the cancel button and the ok button. The ok button is the rightmost one and is the default; i.e.,

return or enter is equivalent to clicking this button. The leftmost button, the cancel button, can be selected via the escape key (esc) or by the command-period combination.

Selecting the ok button (or pressing the enter or return keys) exits the command and returns a list of the items selected. If empty lists are not allowed (the sixth parameter which defaults to true), the ok button will not be available unless at least one item is selected. If empty lists are allowed and nothing is selected when the user chooses ok, an empty list is returned. Canceling the box also exits the command but returns false.

The seventh parameter, initially select, provides a list of items which all appear in the first parameter's list. These items will be selected when the list is first displayed. The names of items in this list (the initial selection list) must match the names of the items in the first parameter's list (the display list) exactly.

NOTE: This command also supports the up and down arrow keys, page up and down, and home and end. In addition, command-arrow key combinations are equivalent to the page up and down keys; for example, command-up arrow is equivalent to the page up key.

Object Database **57** **1.0.0r1**

This group of commands allows you to create files containing generic AppleScript values; i.e., anything that you can set a variable to or return from a script can be stored in an object database.

These commands use the Dictionary Manager, a manager designed for use with text input methods and other text services. This fact limits the size of the data you store to 4K (4096 bytes). If I get enough interest, I'll write my own data storage routines to support unlimited data size (other than the obvious constraints: available memory, the Resource Manager, the Memory Manager, etc.). For more information on the Dictionary Manager consult *Inside Macintosh - Text* chapter 8.

---

create database

Input: create database <file specification>[with maximum key length of <small integer>] [case sensitive searches <boolean>][diacritical sensitive searches <boolean>]
Output: none

This command creates a new database file using the file specification given by the first parameter. The second parameter specifies the maximum length of the keys which are used to retrieve the data. The smaller the keys, the faster the database access will be. If this parameter is not specified, the key can contain up to 4 characters (not including length byte). The last two parameters determine whether keys should be case sensitive and diacritical sensitive respectively. The new file will be created with no records. This command does not open the database.

open database

Input: open database <file specification>[with access permission <open enumeration>]
Output: <integer>
<open enumeration>:
read only/write only/read and write

Use this command to open an existing database file for access. The first parameter specifies the database file to open. The second parameter defines what access rights are available: read only, write only, or read and write. If this parameter is left out, read and write access will be valid.

The command returns a reference number for this open database. Use this reference number in other commands to operate on the database.

---

close database

Input: close database <integer>
Output: none

This command closes the database specified by the reference number in the first parameter. The reference number is invalid after the database is closed.

---

get database information

Input: get database information <integer>
Output: <record> containing {file spec:<file specification>, number of records:<integer>, garbage size:<integer>,maximum key length:<small integer>,case sensitive:<boolean>,diacritical sensitive:<boolean>}

This command returns some basic information about the database specified by the reference number in the first parameter.

The returned record contains the file specification for the database's file, the number of records stored in the database, the amount of wasted bytes (see compact database below), the maximum key length, whether searches are case sensitive, and whether searches are sensitive to diacriticals respectively.

---

find record in database

Input: find record in database <integer>[which has key <string>][of index <integer>]
Output: record containing {key string:<string>,data object:<anything>}

To retrieve an object (record) from the database, use this command. You can use one of two methods to specify which record to retrieve. The first parameter specifies the database containing the object you want. The second parameter specifies the key for the record you want. If you do not include this optional key parameter, you must include the optional index parameter. One and only one of the two optional parameters must be specified. Retrieving

records by index is one-based and allows enumeration of an entire database. The get database information command described above returns the number of records in a database.

The command returns a record containing the key for the record in addition to the record's data. The key is returned to facilitate getting a key for an indexed record.

---

insert data

Input: insert data <anything> into database <integer> with key <string>[using <insert enumeration>]
Output: none
<insert enumeration>:
insert mode/replace mode/forced mode

This command allows you to add records (objects) to an open database. The first parameter is the object you wish to add. This object can be anything that an AppleScript variable can be set to as long as its data does not exceed 4096 bytes. You can store numbers, strings, lists, records, booleans, scripts, etc. in these databases for later retrieval.

The second parameter is the reference number for the database to insert the object into. The third parameter specifies what key to store the object under. This key must contain at most the maximum number of characters allowed for this database as specified when the database was created. The final parameter is optional and indicates how insertions should be handled:

- insert mode: insert the record ONLY if no existing records have the same key according to the search criteria used by find record in database described above.
- replace mode: insert the record ONLY if it replaces an existing record with a matching key.
- forced mode: insert the new record if it replaces an existing one or not.

The default mode is insert mode if the parameter is not passed. If any errors occur during this command, the record is NOT inserted.

---

delete record in database

Input: delete record in database <integer> which has key <string>
Output: none

This command deletes a record from a database. The first parameter is the reference number of the open database, and the second parameter is the key of the record to delete.

---

compact database

Input: compact database <integer>
Output: none

This command compacts the database specified by the reference number in the first parameter. Databases need compaction because when a record is deleted, the space it took up is not "deleted" to save time. If the space of a deleted record was reclaimed during deletion, deletion would require copying on average half of the database's data. The get database information command returns the number of bytes of "garbage" data currently in the database. After compaction, this number will be zero.

Offsets of 40 1.0.0r1

Input: offsets of <string> in <string>][case <case enumeration>]
Output: <list> of <small integers>
<case enumeration>:
insensitive/sensitive

This command returns a list of small integers specifying the locations of a substring in a string. The first parameter is the substring and the second is the target string. The final parameter specifies whether or not case should be ignored. The default value for the optional case parameter is sensitive; i.e., two letters do not match unless they are the same case in addition to being the same letter.

```
offsets of "abc" in "abcdefg…ABCDEFG…" case sensitive
    => {1}
offsets of "abc" in "abcdefg…ABCDEFG…" case insensitive
    => {1,9}        -- the '…' in the above strings is one character (option-;).
```

One important point is that when the routine finds the substring within the string, the search for more instances of the substring begins at the second character in the instance just found; for example:

```
offsets of "aba" in "abababa"
    => {1,3,5}
```

Omit 4 1.0.0r2

Input: omit in <string> at <small integer> for <small integer>
Output: <string>

This command removes the specified characters from the string in the first parameter and returns the resulting string. The characters beginning at the character number specified by the second parameter and going for the number of characters specified by the third parameter are removed. For example,

```
omit in "This line hgad a typo!" at 12 for 1
```

will output the string: "This line had a typo!".

Request Attention 30 1.0.0r2

Input: request attention[message <string>][sound file <file specification>][sound number <small integer>][timeout <integer>]
Output: <boolean>

This command posts a notification request to the user; i.e., it asks the user to switch the script's application to the front for interaction in the standard Macintosh way. The command will not return control to the script until the user has switched to the application (or a timeout has occurred - see the paragraph on the timeout parameter). If the command returns because the user has switched to the script's application, the result will be true. A number of variations of the command are available as for the standard Macintosh notification:

```
request attention
```

This command simply causes the application menu (rightmost menu icon) to blink between the current application's icon and the script's application icon until the user switches to the script's application.

request attention message "This text appears in the alert window which interrupts the user"

In this case, the user sees an alert window appear on his/her screen the next time the operating system sees a chance. The alert window contains the text specified in the string parameter. After the user dismisses the window, the application menu will blink as described above until the user switches to the script's application. For all of the examples below, the alert window can be included also by specifying the message parameter.

request attention[message "hi"] sound number 3

This example interrupts the user by playing the 3rd sound in the system sound file (other sounds besides the ones you see in the Sound control panel exist). It then proceeds as described in one of the first two cases depending on whether the message parameter is there or not. If the sound number is specified as 0, the current system alert sound is played.

request attention[message "hi"]sound file file "HD:sounds:hi joe"[sound number 1]

For this format, the initial sound played is taken from the specified file. The sound number determines which sound from the file to play. If the sound number is left out in this case, the first sound in the file is played. After the sound is played, the notification proceeds as in one of the first two examples depending on whether the message parameter was passed or not.

The timeout parameter allows you to specify the maximum number of seconds you want to wait for the user to respond. If the user has not responded by the end of that period, this command returns false. The command returns true in all other cases except for in error situations where it returns false.

Note that if the script is actually in the foreground when this is called, the notification will still be posted, but it will immediately be fulfilled. If an alert window is requested, however, the alert *will* appear. Consult the **Front Application** command for a technique to avoid this situation.

Set Printer to 13 1.0.1r3

Input: set printer to <string>[named <string>][using <port enumeration>][zone <string>]
Output: none
<port enumeration>:
modem port/printer port

This addition performs one of the main tasks of the chooser application. The first parameter specifies the printer *type* to switch to. The printer type is the actual printer in the case of direct connect printers (e.g., serial printer such as the Imagewriter) and the second and fourth parameters are meaningless. The third parameter, however, only applies to direct connect serial printers. This parameter specifies which port, the modem or printer, the printer is attached to.

set printer to "ImageWriter" using the printer port

For remote printers, the second and fourth parameters are important while the third parameter is meaningless. The type specifies a printer type such as LaserWriter. The name parameter (the second one) then specifies which printer of that type to choose, and the fourth parameter specifies which zone the printer of that name and of that type is located:

set printer to "LaserWriter" named "Second Floor Printer" zone "Second Floor"

Sort 26 1.0.1r6

Input: sort <list>[in <direction enumeration>][order <list>][case sensitivity <boolean>]
Output: <list>
<direction enumeration>:

ascending order/descending order

Provide the sort command with a list of data and it sorts it. The first optional parameter (the second parameter) indicates whether the sorting should be in ascending or descending order. The default is ascending order if this parameter is not provided. This sort order parameter defines the global sort direction. The order parameter is only used when the data are records. See the end of this discussion for more information on sorting records. The final, case, parameter dictates whether case matters during the sorting of strings; if the parameter is left out, case is ignored during string sorting; i.e., the default is case insensitive. I should note that because I wanted to support international sorting, sorting in the case insensitive mode is slower than sorting in the case sensitive mode.

The list can contain numerical or string data but a list of numbers can not contain a string unless the string contains a valid number. A list of strings can always contains numbers because they can be made into a string easily. Here are some examples:

```
sort {1, 5, -1, -10, 4, 300, 3} in descending order
    => {300, 5, 4, 3, 1, -1, -10}

sort {"Heather", "Fred", "Holly", "John", "Peter", "Alex"}
    => {"Alex", "Fred", "Heather", "Holly", "John", "Peter"}
```

Another feature of the sort command is its ability to sort records by a set of "keys" you specify. The keys to use are specified in the final parameter. This final parameter is a list of either: field specifications, or records.

If the list contains field specifications only, each value must be a four character string which represents a field in ALL the records passed in the first list; i.e., all of the records in the first list passed must have this field in common. Unfortunately, the four character code is difficult to determine without having internal knowledge of a record's data structure. Hopefully, some day a way will appear to do this by mere field name instead of the field's four-character designator.

If the list contains records, the records must consist of a field specification and a direction as follows:

```
{field spec:"unam",direction: ascending order}
```

This format allows each separate sort key to use its own direction. In the first case where only field specifications are passed, the global sort direction is used for all keys (see the discussion of the second parameter in the first paragraph above) that do not specify a direction.

When sorting records, the records are ordered by the first field specification given in its sort direction (either the global one or its own). If the data in the first key's field for two different records are equal, the second key (if it exists) is used to sort these two records. The second key may have its own sort direction or it may use the global one. If the second key's field for the records match also, the third key (if it exists) is used. If for two records, the number of passed keys is exhausted, the order of the two records is not specified.

Here is an example: (the line with the "=>" before it is the output of the previous line)

```
set s1 to sharing information
set s2 to {user name:"Heather", machine name:"Greg's Macintosh", zone:""}
set s3 to {user name:"Greg", machine name:"PowerPC 620", zone:""}
sort {s1, s3, s2} order {{field spec:"mnam", direction:ascending order}, {field spec:"unam", direction:descending     order}}
    =>    {{user name:"Heather", machine name:"Greg's Macintosh", zone:""}, {user name:"Greg", machine name:"Greg's
          Macintosh", zone:""}, {user name:"Greg", machine name:"PowerPC 620", zone:""}}
```

The actual sort call above (the fourth line) is equivalent to the following call:

```
sort {s1, s3, s2} order {"mnam", {field spec:"unam", direction:descending order}}
```

This call is equivalent because the default global sort order is ascending, and when you specify just a field specification instead of a record in the order parameter's list, the global sort direction is used for that key. Of course you can make the global sort direction descending instead by simply using the second parameter (in descending order).

Another example of using this extension follows. This example demonstrates a technique for sorting your own records. The symbol before the word "class" («) is typed by using option-\ and the closing symbol (») used after the four character code is typed using option-shift-\.

```
set r1 to {«class name»:"Joe", «class age »:22, «class last»:"Smith"}
set r2 to {«class name»:"Susan", «class age »:22, «class last»:"Jones"}
set r3 to {«class name»:"Fred", «class age »:30, «class last»:"Bell"}
set L to {r1, r2, r3}
sort L order {{field spec:"age ", direction:descending order}, "name"} in ascending order
    ==> {{«class name»:"Fred", «class age »:30, «class last»:"Bell"}, {«class name»:"Joe", «class age »:22, «class
        last»:"Smith"}, {«class name»:"Susan", «class age »:22, «class last»:"Jones"}}
```

For those who are interested, the sorting is performed by the standard quicksort algorithm.

Monitors Osaxen

Depth **14** **1.0.0r2**

Input: depth[of monitor <small integer>]
Output: <record> containing {bits:<integer>, color:<boolean>}

This command returns the bit depth and mode of the specified monitor. If no monitor is specified, the main monitor's bit depth and mode is returned. The monitor specification is by index. The order of the monitors is internal to the system and meaningless to the user. See the **Index Of (monitor)** addition for a way to obtain the index of a desired monitor. The index begins at one and proceeds to the number of monitors. The result is a record containing the current bit depth of the monitor and if it is in color mode (true) or gray scale mode (false).

Does Monitor Support Depth **15** **1.0.0r1**

Input: does monitor support depth[for monitor <small integer>] of <integer>[bit <mode enumeration>]
Output: <boolean>
<mode enumeration>:
colour/gray scale

To check if a specified monitor (if it is not specified, the main monitor is used) supports a bit depth in a mode. If the mode is not specified, color (colour in the command) is assumed. If the monitor supports the bit depth in the mode passed, the command returns true; otherwise, it returns false.

Index of (Monitor) **17** **1.0.0r2**

Input: deepest monitor/main monitor/current monitor
Output: <small integer> containing the index of the specified monitor

To find the index of the main, deepest, or current monitor, use this command:

```
set deepOne to (deepest monitor)
set mainOne to (main monitor)
```

set depth of monitor (current monitor) to 16 bit colour

You can use the output of this command as the optional monitor index in the **Depth**, **Does Monitor Support Depth**, and **Set Depth to** commands described elsewhere in this document.

Number of Monitors **23** **1.0.0r1**

Input: number of monitors
Output: <small integer>

This function returns the number of monitors attached to the local computer.

Set Depth to **1** **1.0.0r2**

Input: set depth[of monitor <small integer>] to <integer>[bit <mode enumeration>]
Output: none
<mode enumeration>:
colour/gray scale

This instruction sets the bit depth and mode of the specified monitor. The monitor is specified by its index. See the description of **Does Monitor Support Depth** for information on the monitor index parameter (the first one here). If the monitor index parameter is left out, the main monitor is used. The second parameter is the bit depth to set the monitor to and the final parameter is the mode: color or gray scale. If the mode parameter is left out, it defaults to color (or colour as it appears in the command):

set depth of monitor (main monitor) to 16 bit colour

Multimedia Osaxen

Audio CD **56** **1.0.0r1**

This group of commands all control an audio CD in a CD-ROM drive. If these extensions do not work with your CD-ROM drive, let me know. Each command accepts an optional "in drive <small integer>" parameter. This value selects which CD-ROM drive to work with. Leaving the parameter out or specifying one uses the first CD-ROM drive in the SCSI chain.

The commands to perform most of these actions were extracted from source code written by James Beninghaus (downloaded from Apple's ftp server). I wrote the AppleScript portion around them in addition to some additional commands which seem to work.

---

play audio CD

Input: play audio CD[in drive <small integer>][starting with track <small integer>]
[using <mode enumeration>]
Output: none
<mode enumeration>:
stereo mode/mono mode

This command allows you to play the audio CD in a CD-ROM drive. The first parameter

specifies which drive to play where the first CD-ROM drive is number 1. If you don't specify a drive, the command defaults to the first drive found in the SCSI chain. The second parameter defines the track to start playing. If you do not include this parameter, the first track will begin playing. The third parameter defines the playback mode: stereo or mono. The default mode is stereo.

---

stop audio CD

Input: stop audio CD[in drive <small integer>][after track <small integer>]
Output: none

This command stops playback of an audio CD in the CD-ROM drive specified by the first parameter. If no drive is specified, the first one is used. The second parameter tells the CD-ROM drive to stop playing once the specified track has stopped playing.

---

pause audio CD

Input: pause audio CD[in drive <small integer>]
Output: none

Use this command to pause playback on the CD-ROM drive specified by the first parameter (default 1).

---

continue audio CD

Input: continue audio CD[in drive <small integer>]
Output: none

This command continues playback of a paused CD-ROM drive specified by the first parameter (default 1).

---

eject CD

Input: eject CD[in drive <small integer>]
Output: none

This command ejects the CD in the CD-ROM drive specified by the first parameter (default 1).

---

number of tracks on audio CD

Input: number of tracks on audio CD[in drive <small integer>]
Output: <small integer>

This command returns the number of tracks on the CD in the CD-ROM drive specified by the first parameter (default 1).

---

track information for audio CD

Input: track information for audio CD[in drive <small integer>] for track <small integer>
Output: <record> containing {mins:<small integer>, secs:<small integer>, frames:<small

integer>}

To determine the duration of a track, use this command. The first parameter specifies which drive the CD is in (default 1). The second parameter is required and defines which track to return the duration of. Each second consists of 75 frames (at least 74 seemed to be the highest value I could get).

---

status of audio CD

Input: status of audio CD[in drive <small integer>]
Output: <record> containing {track:<small integer>, song index:<small integer>, track mins:<small integer>, track secs:<small integer>, track frames:<small integer>, disc mins:<small integer>, disc secs:<small integer>, disc frames:<small integer>}

This command returns the current status of an audio CD in the CD-ROM drive specified by the parameter (default 1). The information returns indicates the current track being played, the index into that track (usually 1), the amount of time into the current track (minutes, seconds, and frames), and the amount of time into the disc (minutes, seconds, and frames).

Number of Sounds **24** **1.0.0r1**

Input: number of sounds[in <file specification>]
Output: <small integer>

This function just returns the number of sounds in the specified file. If no file is passed, the number of sounds in the System file is returned (this will not necessarily agree with the number you count in the Sound control panel).

Play Movie in **10** **1.0.0r1**

Input: play movie in <file specification>[using name <string>][at position <point>][at volume <small integer>]
Output: none

This addition plays the movie (If there can be more than one in a file, then it plays the first one.) in the file specified by the first parameter. The string in the second parameter is the name used for the window where the movie is played. If no name is provided, the name of the file containing the movie is used. The third parameter specifies where on the desktop to place the window. If no position is indicated, the window is centered over the desktop. To just hear the sound track for a movie, specify a position which is off the desktop. The fourth parameter allows playback at any sound volume (0 - 7). The volume change is only during movie playback and the original volume as set in the control panel is left unchanged. For example:

```
play movie in file "HD:Movies:First Mac Ad" using name "1984" at volume 7
```

Clicking the mouse in the close box or pressing any key aborts playback immediately.

Play (Sound) **8** **1.0.0r1**

Input: play <object specification>[in <file specification>][at volume <small integer>]
Output: none

This addition plays the sound specified by the first parameter which is in the file specified by the second parameter at a speaker volume set by the third parameter. If a file is not given (the second parameter), the system file is used instead (the sounds you see in the list in the Sound control panel). If no volume is specified, the current volume is used. The volume change is only temporary. After the sound is played at the desired volume, the previous volume (as set in the sound control panel) is reset.

The first parameter is an object specifier. A sound file can contain multiple sounds (like the system file); therefore, some way is needed to choose which one to play. A sound can be specified in the one of the following standard AppleScript ways: name or absolute position. The name method is obvious while the absolute position method is a little more complicated. The following absolute position methods are valid: every, some, first through tenth, middle, last, and numbered. The numbered methods come in two forms: from the first one and from the last one. Numbering from the first one is simple: 1 is the first one, 2 is the second one, etc. Numbering from the last one goes as follows: -1 is the last one, -2 is the second to last one, etc. Here are some examples:

```
play the fifth sound at volume 3
play the middle sound in file "HD:Sounds"
play the last sound
play sound "meow"
play sound -4
play some sound
play every sound
play the 12th sound
play sound i at volume 7
```

Record Sound to **9** **1.0.0r2**

Input: record sound to <file specification>[named <string>][quality <quality enumeration>][at position <point>]
Output: none
<quality enumeration>:
best/better/good

This command presents the standard sound recording dialog box if it thinks a sound recording device exists. If "OK" is selected, the recorded sound is stored as a sound file at the location specified by the first parameter. The sound itself is named by the optional second parameter. If a name is not provided, the sound is called "new sound." The third parameter specifies the recording quality. If the quality is not specified, better is used. The final parameter is the position to use for the dialog box. The default is (50,50) when this parameter is not supplied.
Networking Osaxen

Choose Link 16 1.0.1r2

Input: choose link[using filter <file specification>][with prompt <string>]
Output: <string> containing the chosen application and its location if a network exists

This addition presents the standard Program linking dialog and returns a string specifying the chosen application. If the machine is not connected to a network, the result will be in the following format:

<port name>:<port type>

The port name is generally the application name and the port type is generally the application's creator code followed by "ep01". If the computer is on a network, the following result is returned:

<port name>:<port type> at <object>:<type>@<zone>

If the user cancels, the result will be the empty string, "".

Choose link accepts two inputs. The file specification must (if passed) specify a file containing a compiled script with the following handler in it:

```
on run {portName,portType,authentReq,netAddr}
	-- look at the parameters above and determine if you want this entry in the list. If you do:
		return true
	-- else
		return false
end run
```

This script can interact with a user and is actually a script object. The only required handler is the run handler. Because it is a script object, values of properties you declare are maintained between calls to it. The following script displays a dialog box for each item in the browser dialog. Each successive dialog box displays 1 then 2 then 3 and so on:

```
property num : 0
on run {s1, s2, s3, s4}
	set num to num + 1
	display dialog num
	return true
end run
```

The number of parameters to the handler can not change (of course the names of them can). The third parameter is a boolean value and the other three are strings. The third parameter, authentReq, specifies whether or not linking to the given application would first require some authentication. The first parameter specifies the name assigned the port, e.g. the application name. The second string contains the port type, e.g. the creator of the application in the first four characters and "ep01" in the last four. The fourth parameter contains a string which specifies the network address of the node in question in standard AppleTalk format. This function should return true if you want the node specified in the parameters to be included in the browser dialog and false if you don't want it to be included.

The script is called continuously until the dialog box is dismissed by selecting "OK" or "Cancel"; i.e., the dialog is continuously checking for nodes. If a new computer is turned on somewhere on the network, it will appear in this dialog box (as soon as it opens some ports). This continuous checking, however, results in the same nodes being passed to the routine over and over again.

If the filter script is not specified, no filtering is performed and all entries appear in the dialog box.

The second parameter is a prompt string which is displayed at the top of the browser dialog box. If the prompt is not specified, the browser's default prompt is used:

Choose a program to link to:

List Links **21** **1.0.0r3**

Input: list links[on <string>][in zone <string>][starting at <small integer>][for <small integer>]
Output: <list>

This command returns the names of all the applications available for linking according to the parameters. This list matches the one which appears in the **Choose Link** dialog box (when no filtering is done). Each item in the list is an application's name as a string. To get an application's signature, call the **Application Info for** command with one of the strings as an input and extract the signature field from the resulting record.

The first parameter allows specification of a machine other than the local one (the default). The second string indicates which zone the machine is in. If the first parameter is left out (the machine), the zone parameter is ignored. The default zone is the local one ("*"). The third parameter provides a starting index into the list of applications to return. The fourth parameter specifies the maximum number of applications to return. These last two parameters allow multiple calls to get a very large list thus avoiding memory problems. The starting index defaults to 0 and the last parameter defaults to 10.

The output list contains an entry for each port fitting the parameter's search description in the following format:

<port name>:<port type>

If an empty string is passed to the zone parameter, the current zone is used just as if the parameter is left out.

List Nodes **11** **1.0.0r2**

Input: list nodes[object <string>][type <string>][zone <string>]
Output: <list> containing strings

This command finds the addresses of all nodes fitting the limitations you specify by the three optional parameters. The object is the name of the network node. The default for object is "=" which means any object. The type specifies what the object is; for example, a Macintosh Quadra 700 is a possible type. The machine type is the string that the machine's "About This Macintosh…" apple menu choice in the Finder displays. The default for this parameter is also "=", meaning all types. The final parameter, the zone, is a string specifying which zones the nodes must be in. The default for this parameter is "*", meaning this zone. The "=" wildcard character is not available for the zone specification. The object and type, however, have an additional wildcard character, "≈" (option-x). This character means that the rest of this string can be anything; therefore, to search for all Macintosh's in the current zone:

list nodes type "Macintosh≈"

The command returns a list containing the network address's of all of the nodes which fit the specified description.

List Zones **12** **1.0.0r2**

Input: list zones
Output: <list> containing strings

This command searches for a bridge and asks it for the names of all the zones on the network (an internet in Apple's terminology). The output list contains strings specifying the names of the zones.

Sharing Information **18** **1.1.0r3**

Input: sharing information
Output: <record> containing {user name:<string>, machine name:<string>, zone:<string>,connection:<string>,AppleTalk enabled:<boolean>}

This command collects the standard sharing information for the local machine and returns it in a record. The user name and the machine name are the strings as entered in the Sharing Setup control panel. The zone name is the zone that the machine is on. If the machine is not on a zone, the string will be empty. The connection field specifies what network connection is selected in the Network control panel (e.g., LocalTalk, EtherTalk Phase 2, etc.). The AppleTalk enabled field contains a boolean value which is true if AppleTalk is on and false if it is off (as set in the Chooser for systems before and including 7.1Pro).
OSA Osaxen

Available Dialects **33** **1.0.0r1**

Input: available dialects
Output: <list> containing <record>s containing information about each dialect

To obtain a list of available dialects, use this command. The result it returns is a list of records. Each record contains its class in addition to the following information about an available dialect: its code, name, language code, and script code. The code provides the identifier for that particular dialect. The **Current Dialect** command described below returns the current dialect's code. The language and script codes specify which language this dialect is localized for. For example, with just the U.S. scripting dialect installed, this command returns the following:

{{class:dialect info, code:129, name:"AppleScript English", language code:0, script code:0}}

Current Dialect **32** **1.0.0r1**

Input: current dialect
Output: <small integer>

This command returns the dialect code for the current dialect in use. The dialect code for U.S. English is 129. You can match this to the dialect code in one of the records returned by **Available Dialects** to get information about the current dialect.
PowerTalk Osaxen

Address of 37 1.1.0r7

Input: address of <file specification>/<string>[user ID <integer>][personal catalog <file specification>
Output: <PowerTalk address>

This command allows a script writer to create a PowerTalk address without user interaction. The command handles three basic types of addresses and each has its own parameter format:

Catalogs appearing in the Catalogs icon

Addresses which appear in a catalog existing in the Catalogs icon (which appears on the user's desktop on the right side) are specified by a path string only:

address of "AppleTalk:Joe's Macintosh"

Personal Catalogs

Addresses which appear in a personal catalog (not including business cards) are specified using a file specification and a record name. Personal catalogs do not allow the PowerTalk equivalent of folders as do some catalogs in the above category. These folders are termed directory nodes in PowerTalk. The file specification locates the personal catalog in which the record exists:

address of "Joe's Macintosh" personal catalog alias "HD:System Folder:Apple Menu Items:Personal Catalog"

Business Cards

Records appearing outside a catalog are termed business cards. Because a business card is a record, it specifies one possible address; therefore, a simple file specification is all that is necessary for these:

address of alias "HD:Desktop Folder:Joe"

The user id parameter allows you to use the **Get User** addition to get a user id and then pass it to this routine. If the parameter is not supplied, the routine attempts to use the local id. If the local id has not been unlocked (the Key Chain), this command returns an error number -1561. To specify guest access, pass a zero as the user id.

The output address is a packed dsspec structure for those who are interested (starting with version 1.1.0).

Choose Address 34 1.1.0r3

Input: choose address[user id <integer>][prompt <string>]
Output: <PowerTalk address>

This extension presents a dialog box allowing the user to select a PowerTalk address through either the standard browsing panel or the find panel. The address selection dialog is a modified version of the AddressOMini sample application provided on the WWDC '93 New Technologies CD (one modification is a minor bug fix). I hope to extend this dialog someday to match the AOCE mailer's dialog for address selection. If anyone has a string to PowerTalk address (PackedDSSpec) conversion utility, I would be very interested in seeing it. I spent a little time trying to write one and then gave up.

The user id parameter allows you to use the **Get User** addition to get a user id and then pass it to this routine. If the parameter is not supplied, the routine attempts to use the local id. If the local id has not been unlocked (the Key Chain), this command returns an error number -1561. To specify guest access, pass a zero as the user id.

The prompt is displayed at the top of the dialog box. If this parameter is left out, a default value (which can be found in the resource fork of this extension of course) is used.

The output address is a packed dsspec structure for those who are interested (starting with version 1.1.0).

Get User **35** **1.0.0r2**

Input: get user[<type enumeration>][guest prompt <string>][local ID prompt <string>][specific ID prompt <string>]
Output: <integer>
<type enumeration>:
local only/specific only/local or guest/guest or local/local or specific/specific or local/guest or specific/specific or guest/of any type

This command presents the standard user authentication dialog box. The type enumeration specifies which types of logins are valid. If the type is not specified, the "local only" option is used. The routine returns the user id of the entered user. If a local identification is requested and the Key Chain has already been opened, the routine simply returns the local id *without* presenting the dialog box. The three prompts are used in the case where a user id of that type is being entered. If you want to simply use guest access, you do not need this command -- pass a zero as the user id to other commands.

Local identification corresponds to the local computer's Key Chain. Specific identities are PowerShare users.

Mail To **36** **1.2.0r5**

Input: mail to <PowerTalk address>[enclosures <file specification>][user ID <integer>][message <text>][subject <string>][cc <PowerTalk address>][bcc <PowerTalk address>]
Output: none

This command sends a file or a text message or both to the specified address and if desired cc's and bcc's it to others also. All three of these parameters which accept PowerTalk addresses (see **Choose Address** and **Address of**) can also accept lists of addresses, but each address in a list must be in the format of a single packed dsspec structure (although each packed dsspec structure, i.e. address, can represent a group and therefore multiple addresses). In other words, each address parameter (recipient, bcc, and cc) can be a list of addresses or a single address. An address is the output of one call to either **Address of** or **Choose Address**.

The enclosures parameter accepts a file specification or a list of file specifications to enclose. If this parameter is not passed, no file is enclosed. The message parameter accepts a string containing the contents of the text to be sent (not limited to 255 characters). If this parameter is not passed, no message is sent. Either a file or a message must be included or an error will

result. In addition, PowerTalk limits file only sends (i.e., no message) to one file. In other words, without a message, the enclosures parameter must specify one file.

The user id parameter allows you to use the **Get User** addition to get a user id and then pass it to this routine. If the parameter is not supplied, the routine attempts to use the local id. If the local id has not been unlocked (the Key Chain), this command returns an error number -1561 along with an appropriate error message. To specify guest access, pass a zero as the user id.

The subject parameter specifies the subject of the PowerTalk letter to be sent. The subject appears in a column of the addressee's in box. If a subject is not specified, the name of the file being sent is used. If no file is being sent or more than one file is being sent, the default subject is currently "Note". This default exists in the resource fork of this extension and is thus easily changeable via ResEdit.
Resources Osaxen

Add Picture **43** **1.0.0r1**

Input: add picture <picture> to <file specification>[id <small integer>][name <string>] [replacing allowed <boolean>]
Output: <small integer>

This command allows you to put pictures into files. The picture will reside in the file's resource fork in the standard PICT format with a resource type of 'PICT'. The first parameter is the actual picture. The second parameter is the file to put the picture in. If this file does not exist at all, it will be created. The file type and creator will be set such that the file is a ResEdit document. If the file does exist but has no resource fork, a resource fork will be added.

The third parameter is optional and specifies the resource id of the picture. Consult the "replacing allowed" parameter description for how this command handles the case where a picture of this id already exists in the file. If the parameter is left out, the command automatically generates a unique resource id for the picture.

The name parameter provides a name for the stored picture. All stored resources have a name, id, and index.

The replacing allowed parameter specifies how the command should handle resource id conflicts. If you pass true for replacing allowed, an existing picture with the specified id will be deleted before the new picture is added. If you pass false (the default), an error will result if a resource of the chosen id already exists. A resource id conflict can only occur if an id is passed to the id parameter.

The command returns the resource id of the picture.

Add Resource **49** **1.0.0r1**

Input: add resource <resource> to <file specification>[id <small integer>][replacing

allowed <boolean>]
Output: <small integer>

This command allows you to add a generic resource to a file. The first parameter is the actual resource. Consult the command, **Extract Resource from**, to see the fields of a record of type resource. The second parameter is the file to put the resource in. If this file does not exist at all, it will be created. The file type and creator will be set such that the file is a ResEdit document. If the file does exist but has no resource fork, a resource fork will be added.

The third parameter is optional and specifies the resource id of the resource. Consult the "replacing allowed" parameter description for how this command handles the case where a resource of this id already exists in the file. If the parameter is left out, the command automatically generates a unique resource id for the resource.

The name within the resource structure is used to name the resource in the file.

The replacing allowed parameter specifies how the command should handle resource id conflicts. If you pass true for replacing allowed, an existing resource with the specified id and of the same type will be deleted before the new resource is added. If you pass false (the default), an error will result if a resource of the chosen id already exists. A resource id conflict can only occur if an id is passed to the id parameter.

The command returns the resource id of the resource.

Add String List **45** **1.0.0r1**

Input: add string list <list> to <file specification>[id <small integer>][name <string>]
[replacing allowed <boolean>]
Output: <small integer>

This command allows you to put string lists into files. The string list will reside in the file's resource fork in the standard STR# format with a resource type of 'STR#'. The first parameter is the actual list of strings. The second parameter is the file to put the string list in. If this file does not exist at all, it will be created. The file type and creator will be set such that the file is a ResEdit document. If the file does exist but has no resource fork, a resource fork will be added.

The third parameter is optional and specifies the resource id of the string list. Consult the "replacing allowed" parameter description for how this command handles the case where a string list of this id already exists in the file. If the parameter is left out, the command automatically generates a unique resource id for the string list.

The name parameter provides a name for the stored string list. All stored resources have a name, id, and index.

The replacing allowed parameter specifies how the command should handle resource id conflicts. If you pass true for replacing allowed, an existing string list with the specified id will

be deleted before the new string list is added. If you pass false (the default), an error will result if a resource of the chosen id already exists. A resource id conflict can only occur if an id is passed to the id parameter.

The command returns the resource id of the string list.

Append aete **50** **1.0.0r1**

Input: append aete <resource> to <resource>
Output: <resource>

This command simply appends one aete resource onto the end of the another and returns the new resource. The suite count is correctly set in the new resource to reflect the total number of suites in the new file. This command is currently not intelligent enough to see if the two suites are the "same" and simply transfer events, classes, comparison operators, and constants into the one suite. This would also involve checking if events, classes, comparison operators, and constants exist in both.

This command is probably pretty useless to the average scriptor but I needed it for the installer.

Extract Picture **42** **1.0.0r3**

Input: extract picture[from <file specification>][index <small integer>][id <small integer>][name <string>]
Output: <picture>

This command allows you to extract pictures from files. The picture must reside in the file's resource fork in the standard PICT format. The first parameter specifies the file containing the picture. If you do not specify a file, the picture will be extracted from the current resource path.

Either none of the three optional parameters or just one of them should be specified. The index parameter allows specification of a picture by its index. For example, specifying an index of three, extracts the third picture from the file. The default index is one; i.e., if all three optional parameters are left out, the first picture will be extracted from the file.

The id parameter allows specifying the picture by its resource id, and the name parameter allows specifying the picture by its name.

The command returns the picture.

Extract Resource **48** **1.0.0r2**

Input: extract resource[from <file specification>] of type <string>[index <small integer>][id <small integer>][name <string>]
Output: <resource> containing {resource name:<string>, resource id:<small integer>, attributes: <small integer>, resource type:<string>, resource data: <string>}

This command allows you to extract a resource of any type from a file. The first parameter

specifies the file containing the resource. If you do not specify a file, the resource will be extracted from the current resource path. The second parameter specifies which type of resource to extract.

Either none of the three optional parameters or just one of them should be specified. The index parameter allows specification of a resource by its index. For example, specifying an index of three, extracts the third resource of the specified type from the file. The default index is one; i.e., if all three optional parameters are left out, the first resource of the specified type will be extracted from the file.

The id parameter allows specifying the resource by its resource id, and the name parameter allows specifying the resource by its name.

The command returns a record of type resource. This record contains fields for the following data:

resource name	The name of the resource in the file
resource id	The id of the resource in the file
attributes	The resource attributes of the resource in the file
resource type	The resource type
resource data	The actual data

Consult *Inside Macintosh* for descriptions of these items.

Extract String List **44** **1.0.0r2**

Input: extract string list[from <file specification>][index <small integer>][id <small integer>][name <string>]
Output: <list>

This command allows you to extract string lists from files. The string list must reside in the file's resource fork in the standard STR# format. The first parameter specifies the file containing the string list. If you do not specify a file, the string list will be extracted from the current resource path.

Either none of the three optional parameters or just one of them should be specified. The index parameter allows specification of a string list by its index. For example, specifying an index of three, extracts the third string list from the file. The default index is one; i.e., if all three optional parameters are left out, the first string list will be extracted from the file.

The id parameter allows specifying the string list by its resource id, and the name parameter allows specifying the string list by its name.

The command returns the list of strings.

List Resource Types in **47** **1.0.0r2**

Input: list resource types in <file>

Output: <list>

This addition returns a list of all the resource types found in the specified file.

Number of Resources in **46** **1.0.0r1**

Input: number of resources in <file specification>[of type <type>]
Output: <small integer>

This function returns the number of resources in the specified file. If the optional parameter is left out, the total number of resources in the file is returned. If a type is specified, the number of resources of that type in the file is returned.