Jon's Commands 2.1

You can always find the latest version on my web page: <http://www.seanet.com/~jonpugh/>

Free for noncommercial use. Contact me for a simple cheap license if you wish to include this scripting addition as part of a software solution that you sell. Contact information is below.

What is Jon's Commands?

Jon's Commands provides added functionality in the form of about 30 commands which can be called from AppleScript. To examine the AppleScript syntax of each of the commands, simply drag the Jon's Commands file onto the Script Editor (or one of the commercial script editors), or use the Open Dictionary command to open Jon's Commands. To install Jon's Commands, simply drop it into the Scripting Additions folder inside your Extensions folder which is in your System Folder.

Is this all the documentation there is?

Of course not. The file "Jon's Commands Reference" is an Apple Guide file. It contains a quick reference of all the commands' syntax, examples and explanations. It is designed to be used in your script editor. This file is a replacement for the SimpleText files I used to use which were becoming cumbersome and difficult to navigate. It contains explanations of each command and example scripts.

You install the "Jon's Commands Reference" guide by placing it, or an alias to it, in the same folder as your script editor. "Jon's Commands Reference" will then appear in the help menu on the right side of your menu bar after you restart the script editor.

This guide is especially useful with Script Debugger, available from Late Night Software, since that editor is scriptable. In all other editors, when you press the Copy button a copy of the current command's complete syntax is placed on the clipboard for you to paste into your script (after a brief switch to the Finder). In Script Debugger, the text will be placed directly into the front script window, without the annoying switch to the Finder and back which is necessary with the other editors. Jon's Commands needs to be installed for the Copy feature to work properly with any editor besides Script Debugger, as does the scriptable Finder.

View the Version Information.

Jon's Commands includes these commands:

- deleteFile -- delete a file, empty folder or a list of files and/or empty folders
- renameFile -- rename a file or folder
- moveFile -- move a file, folder or a list of files and/or folders to another folder
- copyFile -- copy a file, folder or a list of files and/or folders
- sound volume -- get the volume setting
- set sound volume to -- set the volume setting
- clipboard info -- get a list of data on the clipboard
- set the clipboard to -- put data on the clipboard
- the clipboard -- get data from the clipboard
- execute FKEY -- run an FKEY resource
- screen list -- describe the monitor configuration
- set screens to -- change the monitor configuration
- keys pressed -- get a list of pressed keys
- machine environment -- get info about the machine via Gestalt
- play sound -- play sound resources, files and descriptors
- run script resource -- run an 'Scpt' or 'scpt' script resource
- free memory -- return the free memory available
- the ticks -- return the current value of tickCount
- walk folders -- walk folders and run a script on each file
- set cursor to -- provide feedback with the cursor
- AE user interaction level -- control the interaction level
- choose color -- use the color picker to select a color
- keyboard lights -- get and set the lights on the extended keyboard
- snag object -- save an object specifier into a resource file
- alias information -- get information about an alias
- load embedded scripting additions -- load scripting additions from an already open resource file so that you don't need to have them installed to use them
- fileIsBusy -- Returns whether or not a file is busy

Jon's Commands includes these objects:

- picture -- for getting them off the clipboard «class PICT»

- sound -- for getting them off the clipboard & playing them «class snd »
- screen info object -- for describing the monitors
- environment info object -- for describing the machine
- alias info object -- for describing an alias

Jon's Commands includes these coercions:

- string to file specification -- 'TEXT' to 'fss '
- styled text to file specification -- 'STXT' to 'fss '
- international text to file specification -- 'itxt' to 'fss '
- script to anything -- 'scpt' to '****'

Notes for specific commands:

copyFile, deleteFile, moveFile & renameFile:

These commands work on files and lists of files. You can use file references (file "disk:folder:foo"), pathname strings or aliases to specify the files and folders for these commands. In contrast to the Scriptable Finder, these commands run faster, don't put up progress dialogs, batch their operations and use temporary memory when available.

You *cannot* use Finder object references with these commands. You need to coerce them to strings or aliases in order to pass them to these commands.

RenameFile can only change the name of a file or folder. The source is a file or folder reference, alias or pathname and the destination is just the new name. It cannot be a new location.

DeleteFile is permanent and immediate. Be very careful with it. I recommend using "moveFile foo to (path to trash)" if you are at all worried or inclined toward cautiousness. This will only delete empty folders unless the "without safety net" parameter is used. Then this will delete full folders, so be **very very** careful with this parameter. This command also has an option "with unlocking" which allows it to delete locked files, but if this option isn't specified and the "without safety net" is used and there's locked files in a folder, then error 27364 will be returned and only the locked files will not be deleted, which is how the trash works too.

MoveFile only moves files or folders around on the disk they are on. It does not copy between disks. It cannot rename or replace files either. The destination is a folder, which you can specify either with a file reference, an alias or as a pathname string.

CopyFile uses temporary memory if available and tries to copy the file in 1 pass for maximum speed. The source may be either a file or a folder, or list of files and/or folders. The destination may be either a file, folder or list of files and/or folders. If both items are lists, then each item is copied into its corresponding item. Files may be copied to existing or nonexistant files or existing folders. Folder must be copied into existing folders. For matching lists, each pair of items must obey these restrictions. If the source is a list and the destination is not, then it must be an existing folder. Any other destination is illegal.

You cannot replace an existing folder so the replacing parameter is ignored when copying folders. You cannot rename a folder while copying it, so you'll need to use renameFile after the copy completes.

If you try to replace a busy file, the copy will succeed but the busy file will be left in the temporary items folder and will appear in the Rescued Items folder in the trash after the next restart. It will then be deleted the next time you empty the trash.

example:

```
copyFile "disk:folder:foo" to "disk2:folder:bar" with replacing
copyFile "disk:folder:foo" to "disk2:folder:bar" replacing yes
copyFile "disk:folder:foo" to "disk2:folder:bar" without replacing
copyFile "disk:folder:foo" to "disk2:folder:bar" replacing ask
deleteFile "disk:folder:foo"
deleteFile "disk:folder:foo" with unlocking
deleteFile alias "disk:folder:"
deleteFile file "disk:folder:" without safety net
moveFile "disk:folder:foo" to "disk:folder2:"
renameFile alias "disk:folder:foo" to "bar"
copyFile {"disk:folder:foo", alias "disk2:folder:bar:"} to "disk3:folder:"
deleteFile {file "disk:folder:foo", "disk2:folder:bar"}
```

execute FKEY:

This loads the specified FKEY resource and jumps to it. It does not press command-shift-N, so it will not trigger QuicKeys macros or any function which patches things to run at FKEY-like key combinations.

example:

```
execute FKEY "Switch-a-roo"
execute FKEY 5
```

keys pressed:

This returns the GetKey info in the form of a list of key names in string form.

example:

```
(keys pressed) contains "Option"
(keys pressed) = {"Command", "Option"}
```

screen list:

This returns a list of screen info records which describe the monitors attached to the computer. You can specify which screen is first in the returned list by using the "starting with" parameter which takes either "main screen", "deepest screen" or "largest screen". The main screen is the one with the menu bar on it. It will always have 0,0 as its upper left coordinate. The deepest screen is the one with the most colors and closest to the main screen if there are more than one. The largest screen is the one with the most pixels. If no parameter is specified, the main screen is returned as the first item in the list.

A bunch of properties are only returned when Display Manager 2 is present. Display Manager 2 was introduced in System 7.5.3 and is available seperately from Apple for earlier systems.

The screen info object has these properties:

- name - *string* - The name of the screen.
- screen id - *integer* - The unique id of the screen.
- screen size - *point* - The height and width of the screen.
- bounds - *bounding rectangle* - The boundary rectangle for the screen.
- refresh rate - *integer* - The refresh frequency of the screen (DM2 only).
- color depth - *small integer* - The screen color depth setting in bits per pixel.
- in color - *boolean* - If the screen is in color (versus grayscale).
- has menu bar - *boolean* - If this is the main screen which the menu bar is on.
- resolution index - *integer* - The index into the supported resolutions list of the current resolution mode (DM2 only).
- safe resolution - *boolean* - Is this resolution safe or may it require a confirmation dialog (DM2 only)?
- supported resolutions - *list of records* - The possible resolutions that this screen supports (DM2 only).

example:

```
screen list

Returns:
{
    {
        name:"AppleVision 1710AV Display",
        screen id:1,
        bounds:{0, 0, 832, 624},
        screen size:{832, 624},
        color depth:32,
        in color:true,
        has menu bar:true,
        refresh rate:75,
        resolution index:6,
        safe resolution:true,
        supported resolutions:{
            {
                resolution index:1,
                screen size:{640, 480},
                refresh rate:60,
                safe resolution:true,
                name:"640 x 480, 60Hz"
            },
            {
                resolution index:2,
                screen size:{640, 480},
                refresh rate:67,
                safe resolution:true,
                name:"640 x 480, 67Hz"
            },
            {
                resolution index:3,
                screen size:{800, 600},
                refresh rate:60,
                safe resolution:true,
                name:"800 x 600, 60Hz"
            },
            {
                resolution index:4,
                screen size:{800, 600},
                refresh rate:72,
                safe resolution:true,
                name:"800 x 600, 72Hz"
            },
            {
                resolution index:5,
                screen size:{800, 600},
                refresh rate:75,
                safe resolution:true,
                name:"800 x 600, 75Hz"
            },
            {
                resolution index:6,
                screen size:{832, 624},
                refresh rate:75,
                safe resolution:true,
                name:"832 x 624, 75Hz"
            },
            {
                resolution index:7,
                screen size:{1024, 768},
                refresh rate:60,
                safe resolution:true,
                name:"1024 x 768, 60Hz"
            },
            {
                resolution index:8,
                screen size:{1024, 768},
                refresh rate:72,
                safe resolution:true,
                name:"1024 x 768, 72Hz"
            },
            {
                resolution index:9,
                screen size:{1024, 768},
                refresh rate:75,
                safe resolution:true,
                name:"1024 x 768, 75Hz"
            },
            {
                resolution index:10,
                screen size:{1152, 870},
                refresh rate:75,
                safe resolution:true,
                name:"1152 x 870, 75Hz"
            },
            {
                resolution index:11,
                screen size:{1280, 960},
                refresh rate:75,
                safe resolution:true,
                name:"1280 x 960, 75Hz"
            },
            {
                resolution index:12,
                screen size:{1280, 1024},
                refresh rate:75,
                safe resolution:true,
                name:"1280 x 1024, 75Hz"
            }
        }
    }
}

number of items of (screen list)
bounds of item 1 of (screen list starting with largest screen)
```

set screens to:

The "set screens to" command takes a list of screen info records like you get from the "screen list" command. None of the fields are required, but several are highly recommended. Notably, the "screen id" and the "color depth" are guessed at. The "screen id" will be filled in with the main screen if not specified and the Display Manager will guess at a color depth.

Any changes made by this command are forgotten when the system restarts, unless you are running Display Manager 2 or greater. This can be had by installing the Display Enabler (available from Apple) or running System 7.5.3 or later. You may also need to throw away the "Display Preferences" file from your Preferences folder if your system will not remember the display settings across restarts. Any attempt to change the "refresh rate" or "resolution index" of a monitor will fail without Display Manager 2 also. It will throw error 11937 if DM2 is required but not found. It will throw error 11936 if it cannot find a specified monitor.

Example Scripts

Here is a handy script which works well from OSA Menu. What it does is save an applet which will set the screen to the state it was when the applet was created and move the icons to the positions they had then too. Basically, the way you use it, you set up your screen for 832x624 and run this script. Then you set it up for 1024x768 (or whatever) and run it again. From that point on, you only need to run the applets to get back and forth. The applets are required because when running the "set screens to" command from OSA Menu in the Finder, the Finder doesn't get any time to update it's understanding of the screen size and thus disallows the icon repositioning. Running it as an applet allows an idle event to get to the Finder and allow it to update. The applet then uses OSA Menu's handler to run the icon repositioning script in the Finder's layer for maximum speed and no updates.

First off, you'll need to have an applet saved for the first script, since the final script copies that applet and replaces the script in it using "store script". This is kind of tricky, but worth the bother because otherwise you can't create an applet from a script. Take this script, change the alias to point at anything, compile and save it, then change the alias to point at the applet itself and resave the script. Now you can run it and it will put a copy of the handler on the clipboard or you can just copy the handler out of it. Make sure you save it with the settings you want on your script created applets, which in my case is don't stay open and don't show splash screen.

```
on SaveAsApplet(theScript, theFile)
  copyFile alias "Macintosh HD:SaveAs Applet" to file theFile replacing yes
  store script theScript in theFile replacing yes
end SaveAsApplet

on run
  set the clipboard to "on SaveAsApplet(theScript, theFile)
copyFile alias \"" & (path to me as string) & "\" to file theFile replacing yes
store script theScript in theFile replacing yes
end
"
end run
```

Now, we need the script which creates the applet. This one goes in OSA Menu.

```
script iconRestore
  property iconNames : {}
  property iconPositions : {}

  on run
    set cursor to watch cursor -- Jon's Commands
    -- set the icons to their proper places
    repeat with i from 1 to number of items of iconNames
      set n to item i of iconNames
      set p to item i of iconPositions
      try
        tell application "Finder"
          set position of item n of desktop to p
        end tell
      on error
      end try
    end repeat
  end run
end script

script screenRestore
  property screenState : {}
  property restoreScript : ""

  on saveState()
    set cursor to watch cursor -- Jon's Commands
    copy (screen list) to screenState
    tell application "Finder"
      set iconList to every item of desktop
      copy {} to iconRestore's iconNames
      copy {} to iconRestore's iconPositions
      repeat with i in iconList
        copy name of i to end of iconRestore's iconNames
        copy position of i to end of iconRestore's iconPositions
      end repeat
    end tell
    set restoreScript to iconRestore
  end saveState

  on run
    if (keys pressed) contains "Option" then
      display dialog "Update to this configuration?" buttons {"Cancel", "OK"} default button "OK" with icon caution
      saveState()
    else
      -- change the screens
      set screens to screenState
      -- create the file and tell OSA Menu to run it
      set f to (path to temporary items folder as string) & "Temp " & ((the ticks) as string)
      try
        store script restoreScript in file f
        tell application "Finder"
          «event osaMosaM» file f
        end tell
      on error theMsg
        display dialog theMsg with icon 0 buttons "OK" default button "OK"
      end try
      deleteFile f
    end if
  end run
end script

on SaveAsApplet(theScript, theFile, replaceMode)
  -- you need to set this alias to point at any applet
  copyFile alias "Macintosh HD:Scripts:SaveAs Applet" to file theFile replacing replaceMode
  store script theScript in file theFile replacing replaceMode
end SaveAsApplet

on run
```

```
  tell screenRestore to saveState()

  -- Figure out the name based on the main screen size
  set rez to screen size of item 1 of (screenRestore's screenState)
  set screenName to (item 1 of rez as string) & " x " & (item 2 of rez as string)
  try
    -- Set the save as directory, which may not work under 7.5.x depending on the General Controls settings
    set defPath to path to desktop as string
    SetCurrentDir defPath -- Donald's Commands
  on error number -1708 -- event not handled, SetCurrentDir not installed
  end try
  -- Find out where to save
  set theFile to new file with prompt "Save desktop configuration as:" default name screenName
  set theFile to theFile as string
  SaveAsApplet(screenRestore, theFile, yes)
end run
```

Now you can select the script from OSA Menu and it will take a snapshot of your desktop and ask you where to save the applet named with the current size of your screen, like "640x480" or somesuch. I just leave it on the desktop. When you run that applet, it will look at the screen size and if it is different, it will change it to the size it saved and reposition all the icons. If the size is already set, it will put up a dialog asking if you want to restore the state that it saved (this is the default), update to the current setup (useful for when you add or reposition icons on the desktop) or cancel.

Here's a simple script which will be useful if you have already been using the script above and have upgraded Jon's Commands, only to find that the script you've been using no longer works because the screen ids have changed (which happened in AppleVision 1.5.2 regardless). Now that I've figured out how to make them 1 based, they won't change again, but since they changed with Jon's Commands 2.0, you'll need to reset your saved scripts. This script will do that.

When you run this script, it will look for a script application on your desktop with the name of your monitor size. In my case it's "832 x 624". It will slam the current screen settings into this applet so that it will have the proper screen id, along with all the other current settings, and will keep its saved Finder icon positions, allowing it to work again. I find it very handy and have it saved in OSA Menu as "Reset Desktop Script". If you have several different desktop scripts for different monitor resolutions, then you'll need to run this a couple of times.

```
set sl to screen list -- Jon's Commands
set rez to screen size of item 1 of sl
set fn to (item 1 of rez as string) & " x " & (item 2 of rez as string)
tell application "Finder"
  set f to item fn of desktop as alias
end tell
set s to load script f
set s's screenState to sl
store script s in f replacing yes
```

Here's another script which simply toggles the screen size of the main display between 832x624 and 1024x768:

```
set s to item 1 of (screen list)
set si to screen id of s
if screen size of s = {832, 624} then
  set screens to {screen id:si, screen size:{1024, 768}, color depth:16}
else
  set screens to {screen id:si, screen size:{832, 624}, color depth:16}
end if
```

example:

```
set screens to {color depth:256}
set screens to {{screen id:1, screen size:{1024, 768}, color depth:256}, {screen id:2, color depth:4, in color:false}}
```

machine environment:

Machine Environment returns a keyworded record object of common machine information from Gestalt. You can also feed it a Gestalt selector and it will return the long word response. This provides complete access to all of the Gestalt functions that exist. You can use the "check bit" parameter to return a boolean which tells if that bit is on. Bits are numbered from 1 to 32 with number 1 being the lowest or rightmost bit. In this way you can get any information from Gestalt.

That this command will return an error of -5551 if the Gestalt selector code is not installed.

The environment info object provides these properties by default:

- machine type - *string* - The name of the Macintosh model (less specific in recent systems).
- CPU type - *string* - The CPU installed.
- System version - *string* - The current System version.
- FPU - *boolean* - Does this machine have an FPU?
- PowerPC - *boolean* - Whether this machine is a PowerPC.
- Color Quickdraw - *boolean* - Does this machine have color Quickdraw?
- Data Access Manager - *boolean* - Does this machine have the Data Access Manager installed?
- software power off - *boolean* - Does this machine turn off when shutdown?

- logical RAM - *integer* - Amount of RAM available to the system.
- physical RAM - *integer* - Amount of RAM installed in this machine.
- active scripts - *number* - Number of script systems active.
- AppleTalk version - *number* - The current AppleTalk version.
- keyboard - *string* - The name of the main keyboard.
- virtual memory - *boolean* - Is virtual memory on?
- scriptable Finder - *boolean* - Is the scriptable Finder present?
- Open Transport - *boolean* - Is Open Transport installed?
- Open Transport version - *number*
- ethernet address - *string* - The hardware ethernet address (can be "" if hardware is not powered up in PowerBooks). This code can conflict with Eudora. It is best used outside of a Eudora tell block.
- Display Manager version - *number*
- owner name - *string* - The owner name in sharing setup.
- sharing name - *string* - The machine name in sharing setup.

Both this and "keys pressed" use STR# resources if you are inclined to localize them. The machine info is a subset of the available Gestalt selectors. If you have need of others, let me know and I might include them in a future version, although you can get raw information for any gestalt selector directly.

example:

```
machine environment


     {
             machine type:"  Macintosh PowerBook",
             CPU type:"603ev",
             System version:8.1,
             FPU:true,
             PowerPC:true,
             Color Quickdraw:true,
             Data Access Manager:true,
             software power off:true,
             logical RAM:83886080,
             physical RAM:83886080,
             active scripts:1,
             AppleTalk version:60,
             keyboard:"PowerBook Extended",
             virtual memory:false,
             scriptable Finder:true,
             Open Transport:true,
             Open Transport version:1.3,
             ethernet address:"",
             Display Manager version:2.04,
             Jons version:2.04,
             owner name:"Jon Pugh",
             sharing name:"PowPowPowerBook"
     }


scriptable Finder of (machine environment)
machine environment "code"
machine environment "code" check bit 0
```

clipboard info, the clipboard & set the clipboard to:

All of these can manage multiple data types. Styled text is the default data type, with plain text (known as string) as the backup. International text is also supported, as are a lot of other data formats. Almost any resource type can be manipulated in AppleScript as an AEDesc in applications. Pictures and sounds are also popular to move around.

It is now possible to save and restore the complete clipboard by using "the clipboard as record" to get all of the data from the clipboard as a record. As a result of this change, you cannot now put a record on the clipboard, since putting a record on the clipboard now results in each keyworded data item being put on the clipboard seperately. This shouldn't be a problem since it's not too useful to put a record on the clipboard.

There is a problem with using the clipboard in multiple programs. That is, you can only use these commands in the front application. The simplest thing to do is use "activate" first to get the application to the front and then nab or grab the clipboard. Some applications don't seem to check if the scrap has changed while they are running since it only changes when you switch layers in System 7. In this case you can simply activate an app and then activate the original.

Note that the "activate" doesn't really work from scripts run via OSA Menu (and possibly other things which run scripts). This is due to what I call the "send proc bug". OSA Menu can't really fix the problem because it isn't an app, but apps which run scripts can avoid this problem. The result of this bug is that while you can activate an app, it won't read or write its clipboard. Other manifestations of this bug include palettes which stay visible when in the background or hidden in the foreground. The only way to avoid this bug is to run scripts as applets or from applications which don't exhibit this behavior.

If you try to access one of these commands in the background, you'll get error -619 which is a fake Process Manager error. I probably should have used a different number, but it's in use now, so I'm reluctant to change it.

Here's a routine to change the clipboard (I use this in my guide scripts):

```
on slamClipboard(theText)
  tell application "Finder"
    if not frontmost then
      set oldApp to every process whose frontmost = true
      activate
      set the clipboard to theText
      activate oldApp
    else
      set the clipboard to theText
    end if
  end tell
end slamClipboard
```

Here's a routine to send your clipboard to another machine. It uses "login as" and "logout" from the GTQ 1.2 Library.

```
set cookie to login as "Clippy" password "Clippy" -- GTQ 1.2
try
  set theText to the clipboard -- Jon's Commands
  tell application "Finder" of machine "Office in a Box"
    if not frontmost then
      set oldApp to path to frontmost application
      activate
      set the clipboard to theText -- Jon's Commands
      open oldApp
    else
      set the clipboard to theText -- Jon's Commands
    end if
  end tell
  logout cookie -- GTQ 1.2
on error m number n
  logout cookie -- GTQ 1.2
  error m number n
end try
```

Here's the same script in reverse. This one gets the clipboard from another machine.

```
set cursor to watch cursor -- Jon's Commands
set cookie to login as "Clippy" password "Clippy" -- GTQ 1.2
tell application "Finder" of machine "Office in a Box"
  activate
  set theText to the clipboard -- Jon's Commands
end tell
tell application "Finder"
  if not frontmost then
    set oldApp to path to frontmost application
    activate
    set the clipboard to theText -- Jon's Commands
    open oldApp
  else
    set the clipboard to theText -- Jon's Commands
  end if
end tell
logout cookie -- GTQ 1.2
set cursor to arrow cursor -- Jon's Commands
```

example:

```
clipboard info
set foo to the clipboard
set foo to the clipboard as "PICT"
set the clipboard to foo
```

sound volume & set sound volume to:

Both of these have been upgraded to use a number between 0 and 8 for the volume, instead of the older 0-7 range. There's also a new parameter "large range" which increases the resolution to QuickTime's default of 0 to 256.

example:

```
set oldVol to sound volume
set sound volume to 5
set sound volume to oldVol
```

or

```
set oldVol to sound volume with large range
set sound volume to 5 with large range
set sound volume to oldVol with large range
```

play sound:

This can play sound ('snd ') resources installed in the system or the current open files by either name or id number. It can also play sound resources out of files and out of AppleScript variables (typically accessed via the clipboard). You cannot interrupt a sound in progress in this version and the sounds are played synchronously. This command is a small improvement on Donald's sample code from the Language Reference Manual.

example:

```
play sound "I'm sorry Dave..."
```

run script resource:

This matches the AppleScript Run Script command but works on resources instead of files. This allows you to bundle several scripts (typically from other OSA components such as QuicKeys) into a single script file or application. I've used it to place the 'scpt' resources from QuicKeys scripts into an applet and run them from there.

There is a problem with having more than 1 'scpt' resource per file and that is the Script Editor. It finds the script to display by using Get1IndResource('scpt', 1) which can return the wrong resource if you have several. If you are literate with ResEdit you can make sure the proper script resource is last in the file, but that's less certain than the other technique I recommend, which is to change the script resource's type to 'Scpt' which is not recognized by the Script Editor. You need to copy the script resource as hex data and paste it into a new empty 'Scpt' resource. This way allows no conflicts. Another solution is to use my ScriptServer application (available on Info-Mac & gaea) which allows you to create script resources of any type and id. This makes it simple to create a script application which combines AppleScript and QuicKeys scripts. You can even record a QuicKeys script, decompile it with ScriptServer, edit it and recompile it as a piece of a script application.

Yet another option is simply to record a QuicKeys script, copy it and paste the text into your AppleScript script with the "run script foo in QuicKeys" command with foo as your text or a variable with text in it. This works well with simple scripts and doesn't require this osax. Unfortunately, the Run Script 1.1 osax has a memory leak when doing this. Running a compiled script resource is marginally faster too.

You can reference script resources by name or number.

example:

```
run script resource "Flail madly"
```

free memory:

This returns the current value of the function FreeMem. It shows the amount of free memory available in the current application's current heap. Useful for finding memory leaks. You can also specify which heap you would like info about with the constants appZone and sysZone.

example:

```
free memory
free memory sysZone
```

the ticks:

This returns the current value of TickCount. This is useful for timing functions from scripts. Unfortunately, this opens the resource file for Jon's Commands every time and thus affects your timing slightly, making it more useful for large timings where the overhead is less instead of small ones.

This example is a script object which can record a series of times using the ticks. Include it at the beginning of your script and call it's methods.

```
script SmartTimer
  property IMrunning : false
  property startTime : 0
  property endTime : 0
  property timeLog : {}

  on startTimer()
    if IMrunning then
      error "Timer already running"
    end if
    try
      set startTime to the ticks
      set IMrunning to true
    on error theMsg number -1708
      error "You need Jon's Commands installed to use this script object."
    end try
  end startTimer

  on stopTimer()
    if not IMrunning then
      error "Timer is not running"
    end if
    set endTime to the ticks
    set IMrunning to false
    set elapsedTime to endTime - startTime
    set timeLog to timeLog & {elapsedTime}
    return elapsedTime
  end stopTimer

  on averageTime()
    set sum to 0
    set n to number of items of timeLog
    repeat with i in timeLog
      set sum to sum + i
    end repeat
    return sum / n
  end averageTime

  on getLog()
    return timeLog
  end getLog

  on clearLog()
    set timeLog to {}
  end clearLog
end script
```

walk folders:

This makes file processors very simple and fast. It takes a list of files and folders and a script. It then walks through them all and sends each file to an open handler in the script. Note that as of version 1.3.4, walk folders steps backwards through directories (reverse alphabetical order). This changed back to alphabetical order in version 2.1 when I switched to caching a list of aliases instead. This has the benefit of allowing you to modify any aspect of the folders or files without affecting subsequent items.

The "only using files of type" parameter allows you to specify one or more file types that will be operated on. Only files of the types specified will be sent to your script parameter. The types are 4 character strings with the obscure codes you will have to find out about. Normal choices are things like "APPL", "osax" and "TEXT".

In addition, there are 3 boolean parameters, "invisibles", "using files" and "using folders". "invisibles" defaults to true (for compatibility) and causes invisible files to be run through the script. "using files" defaults to true and causes all files to be run through the script. "using folders" defaults to false and causes all folders to be run through the script. You can set them to true or false by using "without invisibles", "with using folders" and/or "without using files".

example:

```
on open (theFiles)
  script foo
    property fileList : {}
    on open (theFile)
      copy theFile as string to end of fileList
      --display dialog theFile as string
      return fileList
    end open
  end script
  set fl to walk folders theFiles with script foo only using files of type {"TEXT", "osas"}
  display dialog (number of items of fl) as string
end open
```

set cursor to:

This allows you to change the current cursor to indicate a busy script. It is problematic since the front application can change it right back, and many do, but it can be very useful nonetheless.

You can use the five standard cursors; arrow cursor, watch cursor, I beam cursor, cross cursor & plus cursor, plus a busy cursor which spins. You can also specify the name or number of a CURS resource or, using the "with cursor list" parameter, an acur resource. Subsequent calls with the "busy cursor" will increment this cursor one step.

The busy cursor keeps an acur and all the CURS resources in memory between calls. The "with cursor list" parameter is only necessary on the first call and will reinitialize the cursor if always supplied (i.e. the cursor will not spin). You can clear this memory by setting the cursor to something other than a busy cursor (i.e. watch cursor or arrow cursor).

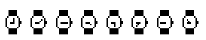
These are the busy cursors available in Jon's Commands, use their names or numbers (names are probably less likely to conflict with other resources):

2128 Counting Fingers

2129 Earth Cursor

2130 Zen Cursor

2131 MPW Cursor


2132 Watch Cursor

This does not support color cursors yet.

example:

```
set cursor to busy cursor with cursor list "Zen Cursor"
repeat
  set cursor to busy cursor
end repeat
set cursor to watch cursor
```

AE user interaction level:

This allows you to get and set the AppleEvent manager's user interaction level. This way you can allow interaction from remote machines or disallow interaction from local scripts. In order for this to work, the program which is doing the interaction must explicitly check if interaction is allowed. Hopefully all scriptable applications do, but reality is seldom so kind.

This returns the current state and then optionally sets it to the new state.

The main drawback with the user interaction level is that you cannot completely turn off user interaction in the current application from a script run by OSA Menu or an application's script menu. Since the lowest interaction level is "AE interact with self" and a script running by OSA Menu or an application itself is sending events to itself, this is always allowed so the app can put up dialogs.

Applications need to support a "never interact" mode explicitly. Apple's PhotoFlash does (shameless plug from one of the authors), but it's only available if you buy a QuickTake camera.

example:

```
set oldLevel to AE user interaction level AE interact with self
AE user interaction level oldLevel
```

choose color:

This function displays the standard color picker and returns an RGB color value. You can specify the prompt and the initial color. RGB colors can be specified via a list of three numbers between 0 and 65535. It's actually a 3 word data value which can be coerced to and from a list. This will throw error 9876 if run on an old 68000 Macintosh without color Quickdraw.

example:

```
choose color
set newColor to choose color "Pick your favorite color:" starting color {0, 0, 0}
```

example script:

```
-- paste an HTML color reference into BBEdit
property lastColor : {35000, 35000, 35000}
set lastColor to choose color "Choose a background color:" starting color lastColor
set s to "#"
set a to "0123456789ABCDEF"
repeat with i in lastColor
  set q to round (i / 65535 * 256)
  if q > 255 then set q to 255 -- limitation of HTML, FF is the brightest
  set h to q div 16
  set l to q mod 16
  set s to s & character (h + 1) of a & character (l + 1) of a
end repeat
tell application "BBEdit 4.0"
  set selected text of window 1 to s
end tell
```

keyboard lights:

This function returns the current setting of the keyboard lights on an ADB extended keyboard and sets them to whatever you specify, which allows you to do a progress indicator which is not obscured by screen savers. Note that this command can override the setting of the caps lock light so that it may not match the actual keyboard setting. Pressing the caps lock key a few times will reset it.

If a keyboard with lights cannot be found, then this command will throw error 23.

The input and return values can be a list of enums or a number. The list has three possible enums in it, num lock light (1), caps lock light (2), and scroll lock light (4). You can use either the enums or an integer combination of the values in parentheses (from 0 to 7). You can also use any other number to signify no change to the lights but that you want the return value to be an integer instead of an enum.

example:

```
keyboard lights {num lock light, caps lock light, scroll lock light} -- turn the lights on & return a list
set oldLights to keyboard lights {} -- turn the lights off & return a list
keyboard lights 6 -- caps lock & scroll lock & return an integer
keyboard lights 0 -- all off & return an integer
set oldLights to keyboard lights -1 -- no change but returns an integer
```

example script:

```
repeat
  repeat with i from 0 to 7
    keyboard lights i
    repeat 10000 times
    end repeat
    if (keys pressed) is not equal to {} then
      keyboard lights 0
      error number -128
    end if
  end repeat
end repeat
```

snag object:

This command is a geeky developer tool mostly of use to people writing applications which want to send a single object specifier to another scriptable application, like the Finder, but don't want to either build the object specifier by hand or load AppleScript for this simple task (typically because of the memory requirements). This command allows them to write the object specfier in their script editor like normal and then, using the "snag object" command, save that object specifier to a file. They can then paste this object specifier resource into their application and build a simple GetData event with it as the direct parameter. The results are returned in the reply and everyone's cruising. See the "Snag Object Demo" folder for working example code.

example:

```
snag object <reference> in <file> id <number> type <string>
```

example script:

```
set f to path to desktop as string
tell application "Finder"
  snag object (creator type of processes whose visible = false) ¬
  in file (f & "Test") id 128 type "obj "
end tell
```

alias information:

This command returns information about aliases. They can be alias variables, alias files, or they can be alias resources in other files. You can specify a specific 'alis' resource by using the id parameter. Without the id parameter, this command looks for id 0 first and then takes the first one it can get via Get1IndResource. If it cannot find an alias, it will throw error 2394.

Normally, this command resolves the alias before it returns any information about it, but this can be troublesome when the alias points to a network server since the server will be mounted, possibly bringing up a login dialog. In order to prevent this, you can specify "without resolving" which will skip this step and the information will be pulled out of the alias without resolving it. Note that this could return incorrect information if the original has been moved or renamed. Also note that the "alias" and "alias was changed" properties of the alias info object are not returned if the alias is not resolved, since these pieces of information require that the resolution succeed.

Beware of passing aliases to aliases to this command when you aren't resolving them. For example:

```
alias information alias "TRex:Desktop Folder:Netscape alias" without resolving
```

This command will return the information for the alias pointing to the Netscape alias file, not the alias **in** the Netscape alias file. You'll need to use a file reference to do that:

```
alias information file "TRex:Desktop Folder:Netscape alias" without resolving
```

Finally, even if the alias was changed when it was resolved, it is not updated on disk. You would need to have the scriptable Finder resolve or replace the alias for that to occur. You can accomplish this most easily simply by getting the "original item" of the alias file. That will cause the Finder to update the alias.

The alias info object has these properties:

- alias zone name - *string* - The zone of the original item.
- alias machine name - *string* - The machine name of the original item.
- alias volume name - *string* - The name of the volume that the original item is from.
- alias path - *string* - The full path of the alias.
- alias - *alias* - The alias itself. This is only present if the alias was resolved.
- alias was changed - *boolean* - Was the alias changed when it was resolved? This is only present if the alias was resolved.

example:

```
alias information <file> id <number> without resolving
```

example script:

```
set foo to (path to apple menu items folder as string) & "Control Panels"
alias information foo
```

load embedded scripting additions:

This command is another geeky developer tool, however this one is for AppleScript application developers who wish to bundle osaxen into their standalone script applications (applets and droplets). This can reduce the installation clutter down to simply installing Jon's Commands instead of a whole slew of scripting additions.

The technique is relatively simple, given an understanding of ResEdit. You simply copy the osax resources, and whatever other resources the osaxen may use (there's no way of telling aside from asking the developer, which you should do anyhow) into your finished applet. Be sure to save this for the last step, since changing the applet with the Script Editor can delete the resources you copied into it. Before you have the resources embedded, make the single "load embedded scripting additions" call as the first line of your script and then you won't need to have those scripting additions installed in the system, only in your applet.

In the case where a scripting addition is installed in the system and the same one (or a different version of the same one) is in the applet, the one installed in the application by this command will be used, solely because it is in the application's event handler table, which is searched before the system's table where all the other scripting additions are.

This command should only be called once because it loads the 'osax' resources into memory, detaches them and installs them in the event handler table. It does not keep any reference to those memory blocks, so they are not reclaimed until the application quits. Calling this multiple times will install multiple copies and use a potentially significant amount of memory.

As I said before, you'll still need to have Jon's Commands installed in the Scripting Additions folder for this to work. It would take a new applet shell (the code that runs your script) to remove this restriction. You should also make sure that you aren't violating scripting addition author's distribution restrictions. For example, Jon's Commands requires a $25 licence to ship it with your commercial or shareware software, but it's free to include with freeware.

There's only one parameter to this command and it should generally be used. It is "only the top resource file used" and restricts the resource loading to only the applet, which is the top resource file (excepting Jon's Commands and AppleScript which are opened on top of the script application calling this command). The parameter is a boolean parameter, so it is accessed via the with/without keywords. The default is without, so you only need to use it with "with". With this command and this parameter, I'm betting that I've got the longest 1 parameter event in all of AppleScript.

This will return error 11934 if there are no 'osax' resources to install and 11935 if there is an 'osax' resource with an invalid name.

examples:

```
load embedded scripting additions
load embedded scripting additions with only the top resource file used
```

fileIsBusy:

This simple command returns whether a file is busy or not. Busy files cannot be opened or deleted. Typically, busy files are open by either the system or another application.

examples:

```
set busyFinder to FileIsBusy alias "Macintosh HD:System Folder:Finder"
```

File Coercions:

- string to file specification
- styled text to file specification
- international text to file specification

These coercions allow you to use strings where a program or osax expects a file specification. Note that the AppleScript class "file" is really an object reference, not simply an FSSpec, although it can be coerced to one. You can create an FSSpec by using this command: set x to "disk:folder:file" as «class fss ». They are particularly easy to implement. With these coercions you don't need to worry about them though.

Script Coercions:

- script to anything

This coercion allows you to coerce a script, as obtained from the "load script" scripting addition, to any type that it can be decompiled into. Normally, this is text, string or styled text.

Version History:

2.1 - official release
2.1b9 - Added individual 'osiz' resources, allowing some commands like "the ticks" to increase in speed by not opening the Jon's Commands resource file. Removed obsolete "finder selection" command since it crashes and everyone's running the scriptable Finder these days.
2.1b8 - Added "large range" parameter to "sound volume" and "set volume to" commands and made them use a range of 0-8 by default instead of 0-7. This could affect scripts, although I doubt it will since the difference between 7 and 8 is pretty minor. With the "large range" parameter enabled, the range is extended from 0 to 256.
2.1b7 - Finally made "walk folders" create a list of aliases and then process it. *This affects script operation!* The files are now processed in alphabetical (for HFS volumes) order. Files can now be renamed or deleted without interfering with the indexing. This is a good thing and makes scripts simpler and more robust. Also added "invisibles" parameter.
2.1b6 - Added "FileIsBusy" command.
2.0.6b6 - NotSoDebugging release for "deleteFile" command. Recompiling appears to have fixed the problems.
2.0.6b5 - Debugging release for "deleteFile" command.
2.0.6b4 - Gave "screen list" and "set monitors to" a larger number of possible screen resolutions, up from 32 to 64.
2.0.6b3 - Made "copyFile" return references to files copied.
2.0.6b2 - Fixed alias path bug in "alias information" command. Still not sure why it stopped working.
2.0.6b1 - Fixed multiple monitor resolution index bug in "screen list". Touched "deleteFile" command for debugging purposes.
2.0.5 - official release
2.0.5b3 - Fixed string parameter parsing in "copyFile" which caused some legal pathnames to be invalid.
2.0.5b2 - Fixed disk custom icon stomping bug in "copyFile" and removed forgotten debugging break which would crash if Macsbug weren't installed and the replacing parameter was used.
2.0.5b1 - Improved string to file coercion a bit more.
2.0.4 - official release
2.0.4b4 - Fixed problem with coercing strings to files. This showed up as errors when running small literal scripts with the "run script" command.
2.0.4b3 - Made "copyFile" command not stomp on specified replacing option when copying a list of files so that it wouldn't ask for every file.
2.0.4b2 - Added "Jons version" to "machine environment". Redid "moveFile" 2.0.3 changes after losing source in a horrible backup accident. Proofed and tweaked a bunch of this document.
2.0.3 - official release
2.0.3b1 - Fixed Finder flag mangling in "moveFile" command when moving items to the desktop.
2.0.2 - limited official release
2.0.2b2 - Added error messages for invalid copyFile parameters. Added record format for "the clipboard" and "set the clipboard to" commands.
2.0.2b1 - Fixed copyFile for "replacing ask" with multiple destinations.
2.0.1 - official release
2.0.1b10 - Enlarged resolution buffer from 15 possible resolutions to 32 in "screen list" and "set screens to" commands. Also added bounds checking since C doesn't provide any. This bug affected 9600s and possibly other machines and caused a crash. Also changed ethernet address in "machine environment" command to return "" instead of "Error" when portable ethernet hardware is unpowered. **This could affect scripts**, although the "Error" string was undocumented and only appeared on a few PowerBooks.
2.0.1b3 to 2.0.1b9 - Debugging thrash for "screen list" command crasher.
2.0.1b2 - Added "Display Manager version" and "Open Transport version" to "machine environment" command. Made "set screens to" report other errors besides the overly popular "Display Manager 2 required" error.
2.0.1b1 - Turned on debugging symbols in all commands.
2.0 - official release
2.0b6 - Removed "with unsafe modes" parameter and added "safe resolution" property instead, as well as adding "resultion index" for more reliable access to the various resolution modes.
2.0b5 - Added "with unsafe modes" to "screen list" command.
2.0b4 - Removed invalid modes and any occurances of a 0 refresh rate from the "supported resolutions" property of "screen list" command. Added SmartTimer to guide file.
2.0b3 - Recompiled "copyFile" and "alias information" commands with MoreFiles 1.4.6, for whatever that is worth. Made "run script resource" return error numbers and messages. Added "safety net" parameter to "deleteFile" command to prevent accidental deletion of full folders. **This affects scripts!** Added script coercion handler. Reversed the order of this change log.
2.0b2 - Extensive modifications to "screen list" and "set screens to" commands to make screen ids 1 based, add actual screen names, refresh rates and lists of possible resolutions. Unfortunately, dealing with the refresh rate requires Display Manager 2, which is included in System 7.5.3 and available separately. If DM2 is not installed, the "screen list" command will not return refresh rate or supported resolutions and the "set screens to" command will error if you try to change the refresh rate.
2.0b1 - Made "alias information" command take an alias descriptor also instead of only alias files. This could affect scripts since an alias to an alias file will now return info about the alias instead of the alias file. Made "machine environment" command return Open Transport status and hardware ethernet address if OT is running. Added "load embedded scripting additions" command. Reworked docs in HTML. Bumped the version up to 2.0 just for the heck of it.
1.8.1 - official release
1.8.1b3 - Shortened the copyFile temp file name so that it couldn't get too long & recompiled with MoreFiles 1.4.3.
1.8.1b2 - Fixed missing watch cursor from animated cursor in "set cursor to" command. Changed a bunch of the ids for resources just in case since they might conflict with application resources when embedding osaxen. Now they're all in wackier ranges. **This could affect scripts!** This could affect the "set cursor to" command in your scripts if (and only if) you refer to cursor lists by number instead of by name. Made "play sound" recognize and follow alias files.
1.8.1b1 - Made "machine environment" recognize PowerBook keyboards and 603e, 603ev and 604e processors.
1.8 - official release
1.8b5 - Made "walk folders" return errors from the script properly (they were all -1753 before).
1.8b4 - Made "machine environment" handle missing machine or owner name safely.
1.8b3 - ++futz
1.8b2 - Futzed with it.
1.8b1 - Added "alias information" command.
1.7.2 - official release
1.7.2b1 - Reworked ordering of "walk folder" execution so that deleting files & folders is possible. Made "deleteFile" delete folders recursively.
1.7.1 - official release
1.7.1b2 - Added this comment for you comment aficionados.
1.7.1b1 - Fixed moveFile for multiple files and put away functionality.
1.7 - official release
1.7b4 - Removed the **** to TEXT coercion because of its lack of usefulness and potential for engendering confusion.
1.7b3 - Added a couple more parameters to the "snag object" command. Removed the Display Enabler since it's too big and available seperately.
1.7b2 - Added "snag object" command.
1.7b1 - Added **** to TEXT coercion.
1.6 - official release
1.6b1 - Added "keyboard lights" command.
1.5.1 - official release
1.5.1b1 - Reverted to 1.4's "finder selection" command because the new sc compiler appears to have changed the object layout so that it doesn't match the Finder's objects anymore and thus doesn't work in 1.5. Also ditched the Display Library as it isn't needed.
1.5 - official release
1.5b3 - included Display Enabler for Display Manager 2.
1.5b2 - Added "with unlocking" parameter to "deleteFile" command so it can delete locked files. Made "set screens to" command persistent when running Display Manager 2. Beefed up the examples and moved them to a new file, "Jon's Commands Notes" to avoid the 32K limit of SimpleText.
1.5b1 - Made everything compile with MPW Pro and the new headers. Made "moveFile" support the Finder's Put Away command. Added "Jon's Commands Reference" guide file to the package. Added "set screens to" command.
1.4 - official release
1.4b4 - Fixed "CPU type" and "FPU" to work properly with PowerPC machines and added "PowerPC" property. Added description and examples for "screen list" command to this document. It was always missing and no one noticed.
1.4b3 - Added "owner name" and "sharing name" properties to "machine information" record.
1.4b2 - Added "choose color" command. Wondered if this change log was worth the bother. It's kind of embarassing, but I guess it's educational. I hope. :) Changed the version to 1.4 since we added a command.
1.3.7b1 - Fixed some comments in the dictionary. Changed the "check bit" parameter of "machine environment" command to be zero based instead of one based, making it compatible with Inside Mac. **This affects scripts!**
1.3.6 - official release
1.3.6b4 - Added "check bit" parameter to "machine environment" command.
1.3.6b3 - Fixed "finder selection" crash when trash is selected. There are probably still problems with selecting PowerTalk or GX items which are not files.
1.3.6b2 - Fixed "walk folders" bug when walking folders; files were fine.
1.3.6b1 - Made replacing parameter on "copyFile" accept yes/no/ask & error when used with directories.
1.3.5 - official release
1.3.5b1 - Fixed "copyFile" bug when copying file to file with a different name.
1.3.4 - official release
1.3.4b4 - Made "copyFile" copy folders too.
1.3.4b3 - Made "copyFile" return better errors when files and directories are missing. Zeroed location when not replacing so that the Finder will put the icon in the first empty spot.
1.3.4b2 - Made "copyFile" fail when copying to a nonexistant directory.
1.3.4b1 - Made "walk folders" command step backwards through folders.
1.3.3 - official release
1.3.3b2 - Fixed "copyFile" problem when replacing a nonexistent file.
1.3.3b1 - Made clipboard commands error when in the background.
1.3.2 - official release
1.3.2b5 - replaced guts of "copyFile" with MoreFiles code. Now it works with drop boxes. Spruced up the readme some more.
1.3.2b4 - honest, this time "copyFile" is fixed. It was FSpDelete that was failing.
1.3.2b3 - made "copyFile" delete the temp file even if FSpExchangeFile fails. Also fixed memory leak in "keys pressed".
1.3.2b2 - added additional acur and CURS resources and spruced up the readme.
1.3.2b1 - added "scriptable Finder" to "machine environment" record.
1.3.1 - official release
1.3.1b1 - made aete unpurgeable to avoid Script Editor dictionary bug.
1.3 - official release
1.3b1 - added "only using files of type" parameter to "walk folders" command.
1.2 - official release
1.2b2 - added "AE user interaction level" function.
1.2b1 - added "set cursor to" command.
1.1.3 - official release
1.1.3b1 - recompiled with debugger breaks off. Ooops.
1.1.2 - official release
1.1.2b1 - made string to fsspec coercion more picky so that it doesn't confuse the "run script" osax.
1.1.1 - official release
1.1.1b3 - fixed error handler in "play sound".
1.1.1b2 - fixed "execute fkey" error handling, twiddled clipboard code to remove debugger calls & clean up the code a bit, fixed "finder selection" when items were on desktop of volumes other than the startup volume & made "walk folders" send an alias instead of a file specification for compatibility with AS 1.0.
1.1.1b1 - fixed "machine environment" property name conflict with "machine" by renaming it "machine type" and actually implemented the "cpu type" property. Also fixed the sample script for "walk folders" above.
1.1 - official release
1.1b3 - added "walk folders" command, made "finder selection" return aliases for AS 1.0 compatibility and made "screen list" work without color QuickDraw.

1.1b2 - fixed bug in "set clipboard to" which could cause changes to the clipboard to be unnoticed and fixed a really minor bug in "execute FKEY" which passed a dirty PC to the FKEY in 24 bit mode.
1.1b1 - added "copyFile" command. Made "moveFile", "deleteFile", "renameFile" & "copyFile" accept lists of files, aliases or pathnames. Added string to file specification coercion.
1.0 - official release.
1.0b5 - added "free memory" and "the ticks" commands.
1.0b4 - added "run script resource" command.
1.0b3 - added 7.1.1 to version check for "finder selection".
1.0b2 - fixed key name in "keys pressed" for PowerBook keyboard.
1.0b1 - first public release

Feel free to write with questions, comments or suggestions.
If you want to send money, please write for current addressing.

Jon Pugh
(206) 675-7356
jonpugh@frostbitefalls.com
http://www.seanet.com/~jonpugh/

Created on Sat, Jan 25, 1997 and last modified on Thu, Aug 5, 1999.