QLab macros v4

Please attribute this work if you share it, and please report any bugs or issues you encounter

This version: 17/01/19

Thanks due to Figure 53, Gareth Fry, Mic Pool and the QLab forum

All of this code has been hand crafted by me – occasionally in response to requests from others – but some of the concepts arose from collective research on the forum

Unless marked otherwise, all macros have been tested with QLab 4.4.1 in Mac OS X 10.12.6.

Hotkeys

Macros that are useful to keep on a hotkey so they can act on selected cues, or in the current location – or will be needed often.

Levels

Level bump

Bump the Master on the Levels tab of the selected Audio, Mic, Video or Fade Cue(s) -2.5dB (it should be obvious how to adjust that number):

```
/cue/selected/level/0/0/- 2.5
```

Make mono

Set crosspoints of selected Audio, Mic or Video Cue(s) for mono routing (crosspoints will be set at 0dB for mono files and -6dB for stereo files):

```
-- Best run as a separate process so it can be happening in the background, as it is quite slow

set qLabMaxAudioChannels to 64

tell application id "com.figure53.QLab.4" to tell front workspace
    repeat with eachCue in (selected as list)
```

```
        try
            if audio input channels of eachCue is 1 then
                repeat with i from 1 to qLabMaxAudioChannels
                    setLevel eachCue row 1 column i db 0
                end repeat
            else if audio input channels of eachCue is 2 then
                repeat with i from 1 to qLabMaxAudioChannels
                    setLevel eachCue row 1 column i db -6
                    setLevel eachCue row 2 column i db -6
                end repeat
            end if
        end try
    end repeat
end tell
```

Times

Adjust Pre Wait

Add 1s to the Pre Wait of the selected cue(s):

```
/cue/selected/preWait/+ 1
```

Adjust Duration

Add 1s to the Duration of the selected cue(s):

```
/cue/selected/duration/+ 1
```

Adjust Post Wait

Add 1s to the Post Wait of the selected cue(s):

```
/cue/selected/postWait/+ 1
```

Effective duration

Set Post Wait(s) of selected Group Cue(s) to effective duration (for those times when a countdown display is useful…); the functionality of this script is now built into QLab for “Timeline” Group Cues:

```
-- The effective duration of a Group Cue is the time it will take for the longest of its children to complete, ie: the longest effective duration of one of its children
-- Each child cue has an effective Pre Wait, which is the cumulative time that must elapse before this child's Pre Wait is triggered
```

```
-- In a "Start first child and go to next cue" Group Cue this effective Pre Wait is the sum of all the Pre Waits and Post Waits that have come before
-- Note that auto-follow cues use their Durations as Post Waits (the actual Post Wait property is ignored); this is the value that must be added to the sum above
-- The effective duration of a single cue that does not auto-follow is the sum of its effective Pre Wait and the longer of its Duration or Post Wait
-- The effective duration of a single cue that does auto-follow is the sum of its effective Pre Wait & Duration

tell front workspace
    repeat with eachCue in (selected as list)
        try
            set eachMode to mode of eachCue
            set longestChildEffectiveDuration to 0
            if eachMode is fire_first_go_to_next_cue then
                set effectivePreWait to 0
                repeat with eachChild in cues of eachCue
                    set eachPre to pre wait of eachChild
                    set effectivePreWait to effectivePreWait + eachPre
                    set eachContinueMode to continue mode of eachChild
                    if eachContinueMode is auto_follow then
                        set eachPost to duration of eachChild
                        set eachChildEffectiveDuration to effectivePreWait + eachPost
                    else
                        set eachDuration to duration of eachChild
                        set eachPost to post wait of eachChild
                        if eachPost > eachDuration then
                            set eachChildEffectiveDuration to effectivePreWait + eachPost
                        else
                            set eachChildEffectiveDuration to effectivePreWait + eachDuration
                        end if
                    end if
                    set effectivePreWait to effectivePreWait + eachPost
                    if eachChildEffectiveDuration > longestChildEffectiveDuration then
                        set longestChildEffectiveDuration to eachChildEffectiveDuration
                    end if
                    if eachContinueMode is do_not_continue then
                        exit repeat -- No point looking at any further children that aren't part of the sequence
                    end if
                end repeat
            end if
            set post wait of eachCue to longestChildEffectiveDuration
        end try
    end repeat
```

```
end tell
```

Fades

Fade in

Create a 5s default fade in of the selected Audio, Mic or Video Cue as the next cue in the cue list (also set the `Master` on the `Levels` tab of the selected cue to `-INF` and copy the Pre Wait time from the selected cue – for use in a “Timeline” Group Cue):

```
set userDuration to 5
set userMinVolume to -120 -- Set what level you mean by "faded out" (you can
adjust this to match the workspace "Min Volume Limit" if necessary)

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        if q type of originalCue is in {"Audio", "Mic", "Video"} then
            set originalCueLevel to originalCue getLevel row 0 column 0
            originalCue setLevel row 0 column 0 db userMinVolume
            set originalPreWait to pre wait of originalCue
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set pre wait of newCue to originalPreWait
            set duration of newCue to userDuration
            newCue setLevel row 0 column 0 db originalCueLevel
            set q name of newCue to "Fade in: " & q list name of originalCue
        end if
    end try
end tell
```

Fade in with follow-on

Create a 5s default fade in of the selected Audio, Mic or Video Cue as the next cue in the cue list (also set the `Master` on the `Levels` tab of the selected cue to `-INF` and create a follow-on); doesn't fire in a cart:

```
set userDuration to 5
set userMinVolume to -120 -- Set what level you mean by "faded out" (you can
adjust this to match the workspace "Min Volume Limit" if necessary)

tell front workspace
    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!
    try -- This protects against no selection (can't get last item of
```

```
(selected as list))
        set originalCue to last item of (selected as list)
        if q type of originalCue is in {"Audio", "Mic", "Video"} then
            set originalCueLevel to originalCue getLevel row 0 column 0
            originalCue setLevel row 0 column 0 db userMinVolume
            set continue mode of originalCue to auto_continue
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set duration of newCue to userDuration
            newCue setLevel row 0 column 0 db originalCueLevel
            set q name of newCue to "Fade in: " & q list name of originalCue
        end if
    end try
end tell
```

Fade out

Create a 5s default fade out (and stop) of the selected Group, Audio, Mic or Video Cue (or target of selected Fade Cue) as the next cue in the cue list (the script does not stop (targeted) Video Cues):

```
set userDuration to 5
set userMinVolume to -120 -- Set what level you mean by "faded out" (you can
adjust this to match the workspace "Min Volume Limit" if necessary)

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        set originalCueType to q type of originalCue
        if originalCueType is in {"Group", "Audio", "Mic", "Video"} then
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set duration of newCue to userDuration
            newCue setLevel row 0 column 0 db userMinVolume
            if originalCueType is not "Video" then
                set stop target when done of newCue to true
            end if
            set q name of newCue to "Fade out: " & q list name of
originalCue
        else if originalCueType is "Fade" then
            set originalCueTarget to cue target of originalCue
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCueTarget
            set duration of newCue to userDuration
            newCue setLevel row 0 column 0 db userMinVolume
            if q type of originalCueTarget is not "Video" then
                set stop target when done of newCue to true
```

```
            end if
            set q name of newCue to "Fade out: " & q list name of
originalCueTarget
        end if
    end try
end tell
```

Build

Create a 5s default fade of the selected Group, Audio, Mic or Video Cue (or target of selected Fade Cue) as the next cue in the cue list, building the `Master` +5dB (the script does nothing if the selected Fade Cue targets a Group Cue):

```
set userDuration to 5
set userLevel to 5
set userKindString to "Build: "

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        set originalCueType to q type of originalCue
        if originalCueType is "Group" then
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set duration of newCue to userDuration
            newCue setLevel row 0 column 0 db userLevel
            set q name of newCue to userKindString & q list name of
originalCue
        else if originalCueType is in {"Audio", "Mic", "Video"} then
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set duration of newCue to userDuration
            set currentLevel to originalCue getLevel row 0 column 0
            newCue setLevel row 0 column 0 db (currentLevel + userLevel)
            set q name of newCue to userKindString & q list name of
originalCue
        else if originalCueType is "Fade" then
            set originalCueTarget to cue target of originalCue
            if q type of originalCueTarget is not "Group" then
                make type "Fade"
                set newCue to last item of (selected as list)
                set cue target of newCue to originalCueTarget
                set duration of newCue to userDuration
                set currentLevel to originalCue getLevel row 0 column 0
                newCue setLevel row 0 column 0 db (currentLevel + userLevel)
                set q name of newCue to userKindString & q list name of
originalCueTarget
```

```
            end if
        end if
    end try
end tell
```

Dip

Create a 5s default fade of the selected Group, Audio, Mic or Video Cue (or target of selected Fade Cue) as the next cue in the cue list, dipping the `Master` -5dB (the script does nothing if the selected Fade Cue targets a Group Cue):

```
set userDuration to 5
set userLevel to -5
set userKindString to "Dip: "

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        set originalCueType to q type of originalCue
        if originalCueType is "Group" then
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set duration of newCue to userDuration
            newCue setLevel row 0 column 0 db userLevel
            set q name of newCue to userKindString & q list name of
originalCue
        else if originalCueType is in {"Audio", "Mic", "Video"} then
            make type "Fade"
            set newCue to last item of (selected as list)
            set cue target of newCue to originalCue
            set duration of newCue to userDuration
            set currentLevel to originalCue getLevel row 0 column 0
            newCue setLevel row 0 column 0 db (currentLevel + userLevel)
            set q name of newCue to userKindString & q list name of
originalCue
        else if originalCueType is "Fade" then
            set originalCueTarget to cue target of originalCue
            if q type of originalCueTarget is not "Group" then
                make type "Fade"
                set newCue to last item of (selected as list)
                set cue target of newCue to originalCueTarget
                set duration of newCue to userDuration
                set currentLevel to originalCue getLevel row 0 column 0
                newCue setLevel row 0 column 0 db (currentLevel + userLevel)
                set q name of newCue to userKindString & q list name of
originalCueTarget
            end if
        end if
```

```
    end try
end tell
```

Force fades

Force selected Fade Cue(s) to completion (the script will also optionally act on Fade Cues within selected Group Cues):

```
set userDuration to 1 -- This is the time the fade(s) will be forced to complete in
set userEnterIntoGroups to true -- Set this to false if you don't want the script to act on Fade Cues within selected Group Cues

tell front workspace
    set cuesToProcess to (selected as list)
    set processedIDs to {}
    set fadeCues to {}
    set originalPreWaits to {}
    set originalDurations to {}
    set originalContinueModes to {}
    set i to 0
    repeat until i = (count cuesToProcess) -- Extract just the Fade Cues
        set eachCue to item (i + 1) of cuesToProcess
        set eachType to q type of eachCue
        if eachType is "Fade" then
            set eachID to uniqueID of eachCue
            if eachID is not in processedIDs then
                set end of fadeCues to eachCue
                set end of processedIDs to eachID
            end if
        else if userEnterIntoGroups is true and eachType is "Group" then
            set cuesToProcess to cuesToProcess & cues of eachCue
        end if
        set i to i + 1
    end repeat
    repeat with eachCue in fadeCues
        stop eachCue
        set end of originalPreWaits to pre wait of eachCue
        set end of originalDurations to duration of eachCue
        set end of originalContinueModes to continue mode of eachCue
        set pre wait of eachCue to 0
        set duration of eachCue to userDuration
        set continue mode of eachCue to do_not_continue
        start eachCue
    end repeat
    delay userDuration + 0.1 -- Give the cue(s) time to complete before resetting to the original variables
    repeat with i from 1 to count fadeCues
        set eachCue to item i of fadeCues
        stop eachCue -- In case of Post Wait…
```

```
        set pre wait of eachCue to item i of originalPreWaits
        set duration of eachCue to item i of originalDurations
        set continue mode of eachCue to item i of originalContinueModes
    end repeat
end tell
```

Force running fades

Force running Fade Cue(s) to completion:

```
set userDuration to 1 -- This is the time the fade(s) will be forced to
complete in

tell front workspace
    set fadeCues to {}
    set originalPreWaits to {}
    set originalDurations to {}
    set originalContinueModes to {}
    repeat with eachCue in (active cues as list) -- Extract just the Fade
Cues
        if q type of eachCue is "Fade" then
            set end of fadeCues to eachCue
        end if
    end repeat
    repeat with eachCue in fadeCues
        stop eachCue
        set end of originalPreWaits to pre wait of eachCue
        set end of originalDurations to duration of eachCue
        set end of originalContinueModes to continue mode of eachCue
        set pre wait of eachCue to 0
        set duration of eachCue to userDuration
        set continue mode of eachCue to do_not_continue
        start eachCue
    end repeat
    delay userDuration + 0.1 -- Give the cue(s) time to complete before
resetting to the original variables
    repeat with i from 1 to count fadeCues
        set eachCue to item i of fadeCues
        stop eachCue -- In case of Post Wait…
        set pre wait of eachCue to item i of originalPreWaits
        set duration of eachCue to item i of originalDurations
        set continue mode of eachCue to item i of originalContinueModes
    end repeat
end tell
```

Crash fades

Crash selected Fade Cue(s) to just before completion, starting their target cues (the script will start

the target cue(s) for the selected Fade Cue(s), load the Fade Cue(s) to just before completion and start them):

```
set userDuration to 1 -- This is the time remaining to which to load the
Fade Cue(s) before starting them

tell front workspace
    repeat with eachCue in (selected as list)
        if q type of eachCue is "Fade" then
            set eachCueTarget to cue target of eachCue
            if running of eachCueTarget is false then
                load eachCueTarget time pre wait of eachCueTarget
                start eachCueTarget
            end if
            stop eachCue -- In case the Fade Cue is a follow-on from its
target
            set eachDuration to ((pre wait of eachCue) + (duration of
eachCue)) -- Include the Pre Wait for effective duration!
            if eachDuration > userDuration then
                load eachCue time eachDuration - userDuration
            end if
            start eachCue
        end if
    end repeat
end tell
```

Prepare fades

Prepare selected Fade Cue(s) for preview by starting their target cues:

```
tell front workspace
    repeat with eachCue in (selected as list)
        if q type of eachCue is "Fade" then
            set eachCueTarget to cue target of eachCue
            if running of eachCueTarget is false then
                preview eachCueTarget
            end if
        end if
    end repeat
end tell
```

Audio

Turn on infinite loop

Turn on infinite loop in selected Audio or Video Cue(s):

```
/cue/selected/infiniteLoop 1
```

Loop last slice

Set last slice of selected Audio or Video Cue(s) to loop:

```
/cue/selected/lastSliceInfiniteLoop 1
```

Set rate

Set playback rate of selected Audio, Video or Fade Cue(s) (also acts on MIDI File Cues); this functionality is now built-in too:

```
set userDefaultToVarispeed to true -- Change this to false if you prefer to
default to no pitch shift

-- Declarations

global dialogTitle
set dialogTitle to "Set playback rate"

-- Main routine

if userDefaultToVarispeed is true then
    set customButtons to {"Cancel", "No pitch shift", "Varispeed"}
else
    set customButtons to {"Cancel", "Varispeed", "No pitch shift"}
end if

set {theRate, theOption} to enterANumberWithCustomButtons("Enter the
playback rate:", "", customButtons, 3, 1)

tell front workspace
    repeat with eachCue in (selected as list)
        try
            set rate of eachCue to theRate
            if theOption is "Varispeed" then
                set pitch shift of eachCue to enabled
            else
                set pitch shift of eachCue to disabled
            end if
        end try
    end repeat
end tell

-- Subroutines

(* === INPUT === *)
```

```
on enterANumberWithCustomButtons(thePrompt, defaultAnswer, theButtons,
defaultButton, cancelButton) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
of (display dialog thePrompt with title dialogTitle ¬
                default answer defaultAnswer buttons theButtons default
button defaultButton cancel button cancelButton)
            try
                set theAnswer to theQuestion as number
            on error
                set theQuestion to ""
            end try
        end repeat
        return {theAnswer, theButton}
    end tell
end enterANumberWithCustomButtons
```

Change target

Change selected Audio Cue's target (keeping start/end times); it will also keep the cue's name, if it has been changed from the default:

```
-- QLab retains slice points within the duration of a new File Target but
resets the start & end times (this script maintains start & end times)

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set selectedCue to last item of (selected as list)
        if q type of selectedCue is "Audio" then
            set currentStart to start time of selectedCue
            set currentEnd to end time of selectedCue
            set currentFileTarget to file target of selectedCue
            if currentFileTarget is not missing value then
                tell application "System Events"
                    set whereToLook to (path of container of
(currentFileTarget as alias)) as alias
                end tell
                set newFileTarget to choose file of type "public.audio" with
prompt "Please select the new File Target:" default location whereToLook
            else
                set newFileTarget to choose file of type "public.audio" with
prompt "Please select the new File Target:"
            end if
            set file target of selectedCue to newFileTarget
            set start time of selectedCue to currentStart
            set end time of selectedCue to currentEnd
```

```
        end if
    end try
end tell
```

Convert to wav

Convert selected Audio Cue's target to wav, if it isn't one already (using the “afconvert” command line utility: can be modified to convert to aiff instead; the script keeps start & end times but resets the cue's name to default; bit depth and sample rate are user-defined within the script):

```
-- QLab retains slice points within the duration of a new File Target but resets the start & end times (this script maintains start & end times)

set userFormat to item 1 of {"wav", "aif"} -- Change this to "item 2" to convert to aiff
set userBitDepth to 16
set userBitRate to 44100

-- Declarations

global dialogTitle
set dialogTitle to "Convert to " & userFormat

-- Prepare some variables

if userFormat is "wav" then
    set acceptableTypes to {"com.microsoft.waveform-audio"}
    set formatString to "WAVE -d LEI"
else if userFormat is "aif" then
    set acceptableTypes to {"public.aifc-audio", "public.aiff-audio"}
    set formatString to "AIFF -d BEI"
else
    return -- Protection against erroneous user modification
end if

-- Convert the cue

tell front workspace
    try -- This protects against no selection (can't get last item of (selected as list))
        set selectedCue to last item of (selected as list)
        if q type of selectedCue is "Audio" then
            set currentFileTarget to file target of selectedCue as alias
            tell application "System Events"
                set currentType to type identifier of currentFileTarget
            end tell
            if currentType is not in acceptableTypes then
                set currentStart to start time of selectedCue
                set currentEnd to end time of selectedCue
                tell application "System Events"
```

```
                    set theContainer to path of container of
currentFileTarget
                    set theExtension to name extension of currentFileTarget
                    if theExtension is "" then
                        set theName to name of currentFileTarget
                    else
                        set theFullName to name of currentFileTarget
                        set theName to text 1 through (-1 - ((length of
theExtension) + 1)) of theFullName
                    end if
                    set newFileTarget to theContainer & theName & "." &
userFormat
                    set fileExists to exists file newFileTarget
                end tell
                if fileExists is true then
                    display dialog "The destination file for the conversion
already exists. What now?" with title dialogTitle with icon 0 ¬
                        buttons {"Cancel", "Replace"} default button
"Replace" cancel button "Cancel"
                end if
                display dialog "Preparing to convert…" with title
dialogTitle with icon 1 ¬
                    buttons {"Cancel", "OK"} default button "OK" cancel
button "Cancel" giving up after 3 -- You have 3s to change your mind
                tell me to do shell script "afconvert -f " & formatString &
userBitDepth & "@" & userBitRate & " " & ¬
                    quoted form of POSIX path of currentFileTarget & " " &
quoted form of POSIX path of newFileTarget
                set file target of selectedCue to newFileTarget
                set start time of selectedCue to currentStart
                set end time of selectedCue to currentEnd
                set q name of selectedCue to "" -- Remove this line if you
don't want to reset the cue name too
                display dialog "Done." with title dialogTitle with icon 1
buttons {"OK"} default button "OK" giving up after 5
            end if
        end if
    end try
end tell
```

Split at waveform cursor

Copy and paste the selected Audio Cue, setting the original to end at the cursor and the copy to start from there; it requires that UI scripting be allowed for QLab (`Accessibility` setting under `Privacy` in System Preferences):

```
-- Only works properly when run as a separate process!

tell application id "com.figure53.QLab.4" to tell front workspace
```

```
    try -- This protects against no selection (can't get last item of
(selected as list))

        set selectedCue to last item of (selected as list)

        if q type of selectedCue is not "Audio" then error -- Need to escape
the whole script

        set startTime to start time of selectedCue
        set endTime to end time of selectedCue

        set splitTime to startTime + ((percent action elapsed of
selectedCue) * (duration of selectedCue)) * (rate of selectedCue)
        -- ###FIXME### As of 4.4.1, "action elapsed" reports differently
between clicking in waveform and loading to time when rate ≠ 1

        if splitTime - startTime ≤ 1.0E-3 or endTime - splitTime ≤ 1.0E-3
then error -- No point splitting if within 1ms of top/tail
        -- "percent action elapsed" reports 99.999% not 100% when a cue is
loaded to its end time, so 1ms is a limit imposed by this rounding error

        stop selectedCue -- Just to make it clearer what's going on

        -- Now, a slightly bodgy way of making sure that focus is not in the
Inspector — so copy/paste works properly — and only one cue selected

        moveSelectionUp
        if last item of (selected as list) is not selectedCue then --
Selected cue was at the top of a cue list!
            moveSelectionDown
        end if

    on error

        return -- Don't go any further…

    end try

end tell

-- Use UI scripting to copy & paste (yuck!)

try
    tell application "System Events" to tell (first application process
whose bundle identifier is "com.figure53.QLab.4")
        -- set frontmost to true -- ###TESTING### Need this line if testing
from Script Editor!
        click menu item "Copy" of menu 1 of menu bar item "Edit" of menu bar
1
        click menu item "Paste" of menu 1 of menu bar item "Edit" of menu
bar 1
```

```
    end tell
on error
    my displayAlert("UI scripting failed!", "You need to adjust your privacy
settings to allow QLab to control your computer…", "critical", {"Cancel",
"OK"}, 2, 1)
    tell application "System Preferences"
        activate
        reveal anchor "Privacy_Assistive" of pane id
"com.apple.preference.security"
    end tell
    return
end try

-- Set the times

tell application id "com.figure53.QLab.4" to tell front workspace
    set copiedCue to last item of (selected as list)
    set end time of selectedCue to splitTime
    set start time of copiedCue to splitTime
end tell

-- Subroutines

(* === ERROR HANDLING === *)

on displayAlert(theWarning, theMessage, theIcon, theButtons, defaultButton,
cancelButton) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        if cancelButton is "" then
            if theIcon is "critical" then -- Triangle with app icon
                display alert theWarning message theMessage as critical
buttons theButtons ¬
                    default button item defaultButton of theButtons
            else if theIcon is "informational" then -- App icon
                display alert theWarning message theMessage as informational
buttons theButtons ¬
                    default button item defaultButton of theButtons
            else if theIcon is "warning" then -- App icon
                display alert theWarning message theMessage as warning
buttons theButtons ¬
                    default button item defaultButton of theButtons
            end if
        else
            if theIcon is "critical" then
                display alert theWarning message theMessage as critical
buttons theButtons ¬
                    default button item defaultButton of theButtons cancel
button item cancelButton of theButtons
            else if theIcon is "informational" then
                display alert theWarning message theMessage as informational
buttons theButtons ¬
```

```
                    default button item defaultButton of theButtons cancel
button item cancelButton of theButtons
            else if theIcon is "warning" then
                display alert theWarning message theMessage as warning
buttons theButtons ¬
                    default button item defaultButton of theButtons cancel
button item cancelButton of theButtons
            end if
        end if
    end tell
end displayAlert
```

Copy slice markers

Copy slice markers of selected Audio or Video Cue to the Clipboard as tab-delimited text:

```
set userHeaderRow to "Slice time" & tab & "Play count" -- Set this to false
if you don't want a header

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set selectedCue to last item of (selected as list)
        set recordTable to my recordToDelimitedText(slice markers of
selectedCue, tab, return)
        if userHeaderRow is not false then
            set recordTable to userHeaderRow & return & recordTable
        end if
        set the clipboard to recordTable as text
    end try
end tell

-- Subroutines

(* === TEXT WRANGLING === *)

on recordToDelimitedText(theRecord, theColumnDelimiter, theRowDelimiter) --
[Shared subroutine]
    set passedTIDs to AppleScript's text item delimiters
    set delimitedList to {}
    set AppleScript's text item delimiters to theColumnDelimiter
    repeat with eachItem in theRecord
        set end of delimitedList to (eachItem as list) as text
    end repeat
    set AppleScript's text item delimiters to theRowDelimiter
    set delimitedText to delimitedList as text
    set AppleScript's text item delimiters to passedTIDs
    return delimitedText
end recordToDelimitedText
```

Update all instances

Copy almost all the scriptable settings from the `Time & Loops` and `Device & Levels` tabs of the Inspector for the selected cue, and then apply them to every Audio Cue in the workspace that references the same file target; the script does not process the “integrated fade envelope toggle”, as it can't copy the fades themselves...

```
-- Best run as a separate process so it can be happening in the background

set userChannelCount to 64 -- Set how many outputs you are using (no point settings levels or gangs beyond this number)
set userLevels to true -- Set this to false if you don't want to update levels
set userGangs to true -- Set this to false if you don't want to update gangs
set userPatch to true -- Set this to false if you don't want to update the Audio Output Patch
set userTimes to true -- Set this to false if you don't want to update start/end times and loop settings
set userSlices to true -- Set this to false if you don't want to update the slice settings (including last slice play count)
set userRate to true -- Set this to false if you don't want to update rate & pitch shift settings

-- Declarations

global dialogTitle, startTime
set dialogTitle to "Update all instances"

-- Find the last Audio Cue in the selection and check it has a valid target, or give up

tell application id "com.figure53.QLab.4"

    tell front workspace

        try
            set selectedCue to last item of (selected as list)
        on error
            return -- No selection
        end try

        if q type of selectedCue is "Audio" then
            set theTarget to file target of selectedCue
        else
            return -- Not an Audio Cue
        end if

        if theTarget is missing value then
            display dialog "The selected Audio Cue does not have a valid target." with title dialogTitle with icon 0 ¬
```

```
                buttons {"OK"} default button "OK" giving up after 5
            return
        end if

        my startTheClock()

        -- Copy the general properties (get them all regardless as only
getting the ones we need would take longer)

        set theID to uniqueID of selectedCue
        set thePatch to patch of selectedCue
        set theStart to start time of selectedCue
        set theEnd to end time of selectedCue
        set thePlayCount to play count of selectedCue
        set theInfiniteLoop to infinite loop of selectedCue
        set theLastSlicePlayCount to last slice play count of selectedCue
        set theLastSliceInfiniteLoop to last slice infinite loop of
selectedCue
        set theSliceRecord to slice markers of selectedCue
        set theRate to rate of selectedCue
        set thePitchShift to pitch shift of selectedCue
        set howManyChannels to audio input channels of selectedCue

        -- Copy the levels

        if userLevels is true then
            set theLevels to {}
            repeat with i from 0 to howManyChannels
                repeat with j from 0 to userChannelCount
                    set end of theLevels to getLevel selectedCue row i
column j
                end repeat
            end repeat
            set theLevelsRef to a reference to theLevels
        end if

        -- Copy the gangs

        if userGangs is true then
            set theGangs to {}
            repeat with i from 0 to howManyChannels
                repeat with j from 0 to userChannelCount
                    set end of theGangs to getGang selectedCue row i column
j
                end repeat
            end repeat
            set theGangsRef to a reference to theGangs
        end if

        -- Find the other instances
```

```
        set allInstances to cues whose broken is false and q type is "Audio"
and file target is theTarget and uniqueID is not theID
        set allInstancesRef to a reference to allInstances
        set countInstances to count allInstancesRef

        repeat with i from 1 to countInstances

            set eachCue to item i of allInstancesRef

            if userLevels is true then
                repeat with j from 0 to howManyChannels
                    repeat with k from 0 to userChannelCount
                        setLevel eachCue row j column k db (item (1 + j *
(userChannelCount + 1) + k) of theLevelsRef)
                    end repeat
                end repeat
            end if

            if userGangs is true then
                repeat with j from 0 to howManyChannels
                    repeat with k from 0 to userChannelCount
                        set storedValue to item (1 + j * (userChannelCount +
1) + k) of theGangsRef
                        if storedValue is missing value then set storedValue
to ""
                        setGang eachCue row j column k gang storedValue
                    end repeat
                end repeat
            end if

            if userPatch is true then
                set patch of eachCue to thePatch
            end if

            if userTimes is true then
                set start time of eachCue to theStart
                set end time of eachCue to theEnd
                set play count of eachCue to thePlayCount
                set infinite loop of eachCue to theInfiniteLoop
            end if

            if userSlices is true then
                set last slice play count of eachCue to
theLastSlicePlayCount
                set last slice infinite loop of eachCue to
theLastSliceInfiniteLoop
                set slice markers of eachCue to theSliceRecord
            end if

            if userRate is true then
                set rate of eachCue to theRate
```

```
                set pitch shift of eachCue to thePitchShift
            end if

            if i < countInstances then -- Countdown timer (and opportunity
to escape)
                my countdownTimer(i, countInstances, "instances")
            end if

        end repeat

        my finishedDialog()

    end tell

end tell

-- Subroutines

(* === OUTPUT === *)

on startTheClock() -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        display dialog "One moment caller…" with title dialogTitle with icon
1 buttons {"OK"} default button "OK" giving up after 1
    end tell
    set startTime to current date
end startTheClock

on countdownTimer(thisStep, totalSteps, whichCuesString) -- [Shared
subroutine]
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeMSS(timeTaken)
    tell application id "com.figure53.QLab.4"
        if frontmost then
            display dialog "Time elapsed: " & timeString & " — " & thisStep
& " of " & totalSteps & " " & whichCuesString & ¬
                " done…" with title dialogTitle with icon 1 buttons
{"Cancel", "OK"} default button "OK" cancel button "Cancel" giving up after
1
        end if
    end tell
end countdownTimer

on finishedDialog() -- [Shared subroutine]
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeNiceT(timeTaken)
    tell application id "com.figure53.QLab.4"
        activate
        display dialog "Done." & return & return & "(That took " &
```

```
timeString & ".)" with title dialogTitle with icon 1 ¬
            buttons {"OK"} default button "OK" giving up after 60
    end tell
end finishedDialog

(* === TIME === *)

on makeMSS(howLong) -- [Shared subroutine]
    set howManyMinutes to howLong div 60
    set howManySeconds to howLong mod 60 div 1
    return (howManyMinutes as text) & ":" & my padNumber(howManySeconds, 2)
end makeMSS

on makeNiceT(howLong) -- [Shared subroutine]
    if howLong < 1 then
        return "less than a second"
    end if
    set howManyHours to howLong div 3600
    if howManyHours is 0 then
        set hourString to ""
    else if howManyHours is 1 then
        set hourString to "1 hour"
    else
        set hourString to (howManyHours as text) & " hours"
    end if
    set howManyMinutes to howLong mod 3600 div 60
    if howManyMinutes is 0 then
        set minuteString to ""
    else if howManyMinutes is 1 then
        set minuteString to "1 minute"
    else
        set minuteString to (howManyMinutes as text) & " minutes"
    end if
    set howManySeconds to howLong mod 60 div 1
    if howManySeconds is 0 then
        set secondString to ""
    else if howManySeconds is 1 then
        set secondString to "1 second"
    else
        set secondString to (howManySeconds as text) & " seconds"
    end if
    set theAmpersand to ""
    if hourString is not "" then
        if minuteString is not "" and secondString is not "" then
            set theAmpersand to ", "
        else if minuteString is not "" or secondString is not "" then
            set theAmpersand to " and "
        end if
    end if
    set theOtherAmpersand to ""
    if minuteString is not "" and secondString is not "" then
```

```
        set theOtherAmpersand to " and "
    end if
    return hourString & theAmpersand & minuteString & theOtherAmpersand &
secondString
end makeNiceT

(* === TEXT WRANGLING === *)

on padNumber(theNumber, minimumDigits) -- [Shared subroutine]
    set paddedNumber to theNumber as text
    repeat while (count paddedNumber) < minimumDigits
        set paddedNumber to "0" & paddedNumber
    end repeat
    return paddedNumber
end padNumber
```

Add files from iTunes

Add selected files from iTunes and name the cues accordingly (the script will also attempt to remove track numbers from the start of the names); remember to bundle the workspace at some point to get the audio files in the right place:

```
set userAttemptToRemoveTrackNumbers to true -- Set this to false if you
don't mind having track numbers in your cue descriptions
set useriTunesSelectionCountLimit to 100 -- Protect against inadvertently
trying to import entire playlists selected in the iTunes sidebar (you can
increase this limit)

-- Declarations

global dialogTitle
set dialogTitle to "Add files from iTunes"

-- Check iTunes is running

tell application "System Events"
    set iTunesIsOpen to count (processes whose name is "iTunes")
end tell
if iTunesIsOpen is 0 then
    display dialog "iTunes is not running." with title dialogTitle with icon
0 buttons {"OK"} default button "OK" giving up after 5
    return
end if

-- Test for an acceptable selection

tell application "iTunes"
    set iTunesSelectionCount to count (selection as list)
end tell
if iTunesSelectionCount is 0 then
```

```
    display dialog "There is no selection in iTunes." with title dialogTitle
with icon 0 buttons {"OK"} default button "OK" giving up after 5
    return
else if iTunesSelectionCount > useriTunesSelectionCountLimit then
    display dialog "The selection in iTunes (" & iTunesSelectionCount & ")
is larger than the limit (" & useriTunesSelectionCountLimit & ")." with
title dialogTitle ¬
        with icon 0 buttons {"OK"} default button "OK" giving up after 10
    return
end if

-- Offer escape hatch

if iTunesSelectionCount is 1 then
    set countString to "file"
else
    set countString to "files"
end if

display dialog "Adding " & iTunesSelectionCount & " selected " & countString
& " from iTunes…" with title dialogTitle with icon 1 ¬
    buttons {"Cancel", "OK"} default button "OK" cancel button "Cancel"
giving up after 5 -- You have 5s to change your mind

-- Get the files

tell application "iTunes"
    set selectedFiles to (location of selection) as list
end tell

-- Get the names

set selectedNames to {}
tell application "System Events"
    repeat with eachItem in selectedFiles
        set end of selectedNames to name of eachItem
    end repeat
end tell

-- Attempt to remove track numbers, as necessary

if userAttemptToRemoveTrackNumbers is true then
    set cleanedNames to {}
    set currentTIDs to AppleScript's text item delimiters
    set AppleScript's text item delimiters to space
    repeat with eachName in selectedNames
        if first character of first text item of eachName is in
"01234567890" then
            set end of cleanedNames to rest of text items of eachName as
text
        else
```

```
            set end of cleanedNames to eachName
        end if
    end repeat
    set AppleScript's text item delimiters to currentTIDs
else
    set cleanedNames to selectedNames
end if

-- Add the files

tell front workspace
    repeat with i from 1 to count selectedFiles
        make type "Audio"
        set newCue to last item of (selected as list)
        set file target of newCue to item i of selectedFiles
        set q name of newCue to item i of cleanedNames
    end repeat
end tell

display dialog "Done." with title dialogTitle with icon 1 buttons {"OK"}
default button "OK" giving up after 5
```

Making

Add Start Cue(s) based on elapsed time

Add Start Cue below the selected cue with Pre Wait based on its elapsed time:

```
tell front workspace

    try -- This protects against no selection (can't get last item of
(selected as list))

        set selectedCue to last item of (selected as list)

        if q type of selectedCue is not "Audio" or running of selectedCue is
false then error -- Need to escape the whole script

        set elapsedTime to (percent action elapsed of selectedCue) *
(duration of selectedCue)
        -- ###FIXME### As of 4.4.1, "action elapsed" reports differently
between clicking in waveform and loading to time when rate ≠ 1

        make type "Start"
        set newCue to last item of (selected as list)
        set pre wait of newCue to elapsedTime

        set selected to selectedCue
```

```
    end try

end tell
```

Add Start Cue(s)

Add triggers to the end of the Main Cue List for cues you have selected in another cue list, and then switch back:

```
set userCueList to "Main Cue List" -- Use this to specify the name of the
cue list that receives the Start Cue(s)

-- Declarations

global dialogTitle
set dialogTitle to "Add Start Cues"

-- Main routine

tell front workspace
    try -- Check destination cue list exists
        set startCueList to first cue list whose q name is userCueList
    on error
        display dialog "The destination cue list \"" & userCueList & "\"
does not exist." with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK"
        return
    end try
    set startingSelection to selected
    set startingCueList to current cue list
    set originalCueList to q name of current cue list
    if originalCueList is not userCueList then
        set selectedCues to (selected as list)
        set current cue list to startCueList
        repeat with eachCue in selectedCues
            make type "Start"
            set newCue to last item of (selected as list)
            set cue target of newCue to eachCue
            set nameString to "Start"
            if originalCueList is not "" then
                set nameString to nameString & " | " & originalCueList
            end if
            set eachNumber to q number of eachCue
            if eachNumber is not "" then
                set nameString to nameString & " | Q " & eachNumber
            end if
            set eachName to q list name of eachCue
            if eachName is not "" then
                set nameString to nameString & " | " & eachName
            end if
```

```
            set q name of newCue to nameString
        end repeat
        delay 1 -- Let you see it
        set selected to startingSelection
        set current cue list to startingCueList
    end if

end tell
```

Arm

Create Arm Cue targeting selected cue as previous cue; doesn't fire in a cart:

```
tell front workspace
    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        make type "Arm"
        set newCue to last item of (selected as list)
        set cue target of newCue to originalCue
        set targetName to q list name of originalCue
        if targetName is "" then
            set targetName to q display name of originalCue
        end if
        set q name of newCue to "Arm: " & targetName
        set originalCueIsIn to parent of originalCue
        if parent of newCue is originalCueIsIn then -- Only reorder the cues
if they are in the same group/cue list
            set originalCueID to uniqueID of originalCue
            set newCueID to uniqueID of newCue
            move cue id originalCueID of originalCueIsIn to after cue id
newCueID of originalCueIsIn
        end if
    end try
end tell
```

Disarm

Create Disarm Cue targeting selected cue as next cue; doesn't fire in a cart:

```
tell front workspace
    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
```

```
        make type "Disarm"
        set newCue to last item of (selected as list)
        set cue target of newCue to originalCue
        set targetName to q list name of originalCue
        if targetName is "" then
            set targetName to q display name of originalCue
        end if
        set q name of newCue to "Disarm: " & targetName
    end try
end tell
```

Go ahead make MIDI

Make a series of MIDI Cues which increment the parameter you specify; doesn't fire in a cart:

```
-- Best run as a separate process so it can be happening in the background

-- This script can optionally convert note numbers in MIDI Cues to note
names: use the variables below to set the behaviour

set userMIDIPatch to false -- Set to false to use QLab default for the cues
made, or set to 1-8 for MIDI patch 1-8
set userMinimumTimeBetweenCues to 0.01 -- How long to allow between MIDI
Cues when sending clusters (eg: Bank/PC)?
set userHowManyPossible to 1000 -- Limit series with increment of 0 to this
many cues!
set userMIDIConversionMode to item 3 of {"Numbers", "Names", "Both"} -- Use
this to choose how MIDI note values will be included in cue names
set userNote60Is to item 1 of {"C3", "C4"} -- Use this to set which note
name corresponds to note 60

-- Explanations

set theExplanation to "This script will create a series of MIDI Cues which
increment the parameter you specify.

For example, you can generate quickly a series of Program Changes for
recalling scenes on a mixing desk, " & ¬
    "or a series of Note Ons for triggering a sampler or some other device."

-- Declarations

global dialogTitle
set dialogTitle to "Go ahead make MIDI"

global userMIDIConversionMode, noteNames, octaveConvention
set noteNames to {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A",
"A#", "B"}
if userNote60Is is "C3" then
    set octaveConvention to -2
```

```
else
    set octaveConvention to -1
end if

set kindChoices to {"Note On: fixed note number, incrementing velocity",
"Note On: fixed velocity, incrementing note number", ¬
    "Control Change: fixed controller number, incrementing value", "Control
Change: fixed value, incrementing controller number", "Program Change", ¬
    "Bank Select (Coarse) / Program Change, ie: CC00 + Program Change"}
set kindCommands to {"Note On", "Note On", "Control Change", "Control
Change", "Program Change", "CC00/PC"}
set kindByteOne to {"fixed", "variable", "fixed", "variable", "variable",
"variable"}
set kindByteTwo to {"variable", "fixed", "variable", "fixed", "", ""}
set kindQuestionHeader to {"Making Note On cues with incrementing
velocity.", "Making Note On cues with incrementing note number.", ¬
    "Making Control Change cues with incrementing value.", "Making Control
Change cues with incrementing controller number.", ¬
    "Making Program Change cues.", "Making Bank Select (Coarse) / Program
Change cues."}
set kindQuestionOne to {"Enter the fixed note number:", "Enter the note
number for the first cue:", "Enter the fixed controller number:", ¬
    "Enter the controller number for the first cue:", "Enter the Program
Change number for the first cue:", ¬
    "Enter the combined Bank Select (Coarse) / Program Change number for the
first cue (eg: for Bank 2 Program 3 enter 2 * 128 + 3 = 259):"}
set kindQuestionTwo to {"Enter the velocity for the first cue:", "Enter the
fixed velocity:", "Enter the value for the first cue:", "Enter the fixed
value:", "", ""}
set kindQuestionInc to "Enter the increment:"
set kindQuestionMany to "Enter the number of cues to create (maximum "
set channelChoices to {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16}

-- Find out what we are doing

tell application id "com.figure53.QLab.4" to tell front workspace

    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!

    set theKind to my pickFromList(kindChoices, theExplanation & return &
return & "Choose which kind of series would you like to make:")

    repeat with i from 1 to count kindChoices
        if theKind is item i of kindChoices then
            set commandType to item i of kindCommands
            if commandType is "CC00/PC" then
                set byteOneMax to 127 * 128 + 127
            else
                set byteOneMax to 127
```

```
            end if
            set byteOne to item i of kindByteOne
            set byteTwo to item i of kindByteTwo
            set questionHeader to item i of kindQuestionHeader
            set questionOne to item i of kindQuestionOne
            set questionTwo to item i of kindQuestionTwo
            set questionInc to kindQuestionInc
            set questionMany to kindQuestionMany
            exit repeat
        end if
    end repeat

    set theChannel to my pickFromList(channelChoices, questionHeader &
return & return & "Choose the MIDI channel for the cues:")

    set byteOneStart to my
enterANumberWithRangeWithCustomButton(questionHeader & return & return &
questionOne, "", ¬
        0, true, byteOneMax, true, true, {}, "OK")
    set currentByteOne to byteOneStart
    set byteOneIncrement to 0

    if byteOne is "variable" then
        set byteOneIncrement to my
enterANumberWithRangeWithCustomButton(questionHeader & return & return &
questionInc, "", ¬
            -byteOneStart, true, (byteOneMax - byteOneStart), true, true,
{}, "OK")
        if byteOneIncrement < 0 then
            set howManyPossible to 1 - (round (byteOneStart /
byteOneIncrement) rounding up)
        else if byteOneIncrement > 0 then
            set howManyPossible to 1 + (round ((byteOneMax - byteOneStart) /
byteOneIncrement) rounding down)
        else
            set howManyPossible to userHowManyPossible
        end if
    end if

    if byteTwo is not "" then
        set byteTwoStart to my
enterANumberWithRangeWithCustomButton(questionHeader & return & return &
questionTwo, "", ¬
            0, true, 127, true, true, {}, "OK")
    else
        set byteTwoStart to 0
    end if
    set currentByteTwo to byteTwoStart
    set byteTwoIncrement to 0

    if byteTwo is "variable" then
```

```
        set byteTwoIncrement to my
enterANumberWithRangeWithCustomButton(questionHeader & return & return &
questionInc, "", ¬
            -byteTwoStart, true, (127 - byteTwoStart), true, true, {}, "OK")
        if byteTwoIncrement < 0 then
            set howManyPossible to 1 - (round (byteTwoStart /
byteTwoIncrement) rounding up)
        else if byteTwoIncrement > 0 then
            set howManyPossible to 1 + (round ((127 - byteTwoStart) /
byteTwoIncrement) rounding down)
        else
            set howManyPossible to userHowManyPossible
        end if
    end if

    set howMany to my enterANumberWithRangeWithCustomButton(questionHeader &
return & return & questionMany & howManyPossible & "):", "", ¬
        1, true, howManyPossible, true, true, {}, "OK")

    set customNaming to text returned of (display dialog questionHeader &
return & return & ¬
        "Enter a base string for naming the cues, or press return to use the
default naming option:" with title ¬
        dialogTitle default answer "" buttons {"Cancel", "OK"} default
button "OK" cancel button "Cancel")
    if customNaming is not "" then
        if customNaming does not end with " " and (byteOneIncrement is not 0
or byteTwoIncrement is not 0) then
            set customNaming to customNaming & " "
        end if
    end if

    -- Do it

    display dialog "One moment caller…" with title dialogTitle with icon 1
buttons {"OK"} default button "OK" giving up after 1

    if commandType is not "CC00/PC" then

        set theCommand to commandType

        repeat howMany times

            make type "MIDI"
            set newCue to last item of (selected as list)
            if userMIDIPatch is not false then
                set patch of newCue to userMIDIPatch
            end if
            set channel of newCue to theChannel
            set byte one of newCue to currentByteOne
            set nameString to "Channel " & theChannel & " | " & theCommand &
```

```
" | "
            if theCommand is "Note On" then
                set command of newCue to note_on
                set byte two of newCue to currentByteTwo
                set nameString to nameString & my
convertMIDINoteValues(currentByteOne, currentByteTwo)
            else if theCommand is "Control Change" then
                set command of newCue to control_change
                set byte two of newCue to currentByteTwo
                set nameString to nameString & currentByteOne & " @ " &
currentByteTwo
            else if theCommand is "Program Change" then
                set command of newCue to program_change
                set nameString to nameString & currentByteOne
            end if
            if customNaming is "" then
                set q name of newCue to nameString
            else if byteOneIncrement is not 0 then
                set q name of newCue to customNaming & currentByteOne
            else if byteTwoIncrement is not 0 then
                set q name of newCue to customNaming & currentByteTwo
            else
                set q name of newCue to customNaming
            end if
            set currentByteOne to currentByteOne + byteOneIncrement
            set currentByteTwo to currentByteTwo + byteTwoIncrement

        end repeat

    else

        repeat howMany times

            set currentBank to currentByteOne div 128
            set currentProgram to currentByteOne mod 128

            make type "Group"
            set groupCue to last item of (selected as list)
            set mode of groupCue to timeline

            make type "MIDI"
            set newCue to last item of (selected as list)
            if userMIDIPatch is not false then
                set patch of newCue to userMIDIPatch
            end if
            set channel of newCue to theChannel
            set command of newCue to control_change
            set byte one of newCue to 0
            set byte two of newCue to currentBank
            set q name of newCue to "Channel " & theChannel & " | Control
Change | 0 @ " & currentBank
```

```
            set newCueID to uniqueID of newCue
            move cue id newCueID of parent of newCue to end of groupCue

            make type "MIDI"
            set newCue to last item of (selected as list)
            if userMIDIPatch is not false then
                set patch of newCue to userMIDIPatch
            end if
            set channel of newCue to theChannel
            set command of newCue to program_change
            set byte one of newCue to currentProgram
            set q name of newCue to "Channel " & theChannel & " | Program
Change | " & currentProgram
            set newCueID to uniqueID of newCue
            move cue id newCueID of parent of newCue to end of groupCue
            set pre wait of newCue to userMinimumTimeBetweenCues

            if customNaming is "" then
                set q name of groupCue to "Channel " & theChannel & ¬
                    " | Bank | " & currentBank & " | Program Change | " &
currentProgram & " | Combined value: " & currentByteOne
            else if byteOneIncrement is not 0 then
                set q name of groupCue to customNaming & currentByteOne
            else
                set q name of groupCue to customNaming
            end if

            set currentByteOne to currentByteOne + byteOneIncrement

        end repeat

    end if

    display dialog "Done." with title dialogTitle with icon 1 buttons {"OK"}
default button "OK" giving up after 60

end tell

-- Subroutines

(* === INPUT === *)

on enterANumberWithRangeWithCustomButton(thePrompt, defaultAnswer, ¬
    lowRange, acceptEqualsLowRange, highRange, acceptEqualsHighRange,
integerOnly, customButton, defaultButton) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
of (display dialog thePrompt with title dialogTitle ¬
                default answer defaultAnswer buttons (customButton as list)
```

```
& {"Cancel", "OK"} default button defaultButton cancel button "Cancel")
            if theButton is customButton then
                set theAnswer to theButton
                exit repeat
            end if
            try
                if integerOnly is true then
                    set theAnswer to theQuestion as integer -- Detects non-
numeric strings
                    if theAnswer as text is not theQuestion then -- Detects
non-integer input
                        set theQuestion to ""
                    end if
                else
                    set theAnswer to theQuestion as number -- Detects non-
numeric strings
                end if
                if lowRange is not false then
                    if acceptEqualsLowRange is true then
                        if theAnswer < lowRange then
                            set theQuestion to ""
                        end if
                    else
                        if theAnswer ≤ lowRange then
                            set theQuestion to ""
                        end if
                    end if
                end if
                if highRange is not false then
                    if acceptEqualsHighRange is true then
                        if theAnswer > highRange then
                            set theQuestion to ""
                        end if
                    else
                        if theAnswer ≥ highRange then
                            set theQuestion to ""
                        end if
                    end if
                end if
            on error
                set theQuestion to ""
            end try
        end repeat
        return theAnswer
    end tell
end enterANumberWithRangeWithCustomButton

on pickFromList(theChoice, thePrompt) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        choose from list theChoice with prompt thePrompt with title
dialogTitle default items item 1 of theChoice
```

```
        if result is not false then
            return item 1 of result
        else
            error number -128
        end if
    end tell
end pickFromList

(* === TEXT WRANGLING === *)

on configureNoteNameString(noteNumber) -- [Shared subroutine]
    set theOctave to (noteNumber div 12) + octaveConvention
    set theNote to item (noteNumber mod 12 + 1) of noteNames
    set noteNameString to theNote & theOctave
    return noteNameString
end configureNoteNameString

on convertMIDINoteValues(noteNumber, noteVelocity) -- [Shared subroutine]
    if userMIDIConversionMode is "Numbers" then
        return noteNumber & " @ " & noteVelocity
    else if userMIDIConversionMode is "Names" then
        return my configureNoteNameString(noteNumber) & " @ " & noteVelocity
    else if userMIDIConversionMode is "Both" then
        return noteNumber & " @ " & noteVelocity & " | " & my
configureNoteNameString(noteNumber)
    end if
end convertMIDINoteValues
```

Add a crashable wait cue

Add some ugly cues to make it possible to crash ahead before the next domino has toppled; or, looking at it another way, if you don't take the cue it will GO anyway when the timer (5s) runs out…; doesn't fire in a cart:

```
set userDuration to 5

tell front workspace
    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        set continue mode of originalCue to auto_continue
        make type "Start"
        set newStartCue to last item of (selected as list)
        set cue target of newStartCue to current cue list
        set pre wait of newStartCue to userDuration
        set q name of newStartCue to "     stop…"
        make type "Stop"
        set newStopCue to last item of (selected as list)
```

```
        set cue target of newStopCue to newStartCue
        set q name of newStopCue to "     …carry on"
        set continue mode of newStopCue to auto_continue
    end try
end tell
```

Batch adjusting

Toggle arming

Toggle arming of selected cue(s) (the script will arm/disarm the current cue list if no cues are selected):

```
tell front workspace
    set selectedCues to (selected as list)
    if (count selectedCues) is 0 then -- If no cues are selected arm/disarm
the current cue list
        set armed of current cue list to not armed of current cue list
    else
        repeat with eachCue in reverse of selectedCues -- Reversed so as to
do a Group Cue's children before it
            set armed of eachCue to not armed of eachCue
        end repeat
    end if
end tell
```

Batch arm/disarm

Arm, disarm or toggle arming of all cues in the workspace whose name contains the string you enter (not case-sensitive); the string is copied to the Clipboard for you to paste in next time:

```
set userDefaultSearchString to "REHEARSAL" -- Use this to specify the
default search string

-- Declarations

global dialogTitle
set dialogTitle to "Batch disarm"

-- Get the search string

set {theText, theButton} to {text returned, button returned} of (display
dialog ¬
    "Arm/disarm cues whose name contains (return an empty string to
cancel):" with title dialogTitle with icon 1 ¬
    default answer userDefaultSearchString buttons {"Toggle", "Arm",
"Disarm"} default button "Disarm")
```

```
-- Check for cancel

if theText is "" then
    error number -128
end if

-- Copy the search string to the Clipboard and arm/disarm the cues

set the clipboard to theText as text

tell front workspace
    set foundCues to cues whose q list name contains theText
    set foundCuesRef to a reference to foundCues
    repeat with eachCue in reverse of foundCuesRef -- Reversed so as to do a
Group Cue's children before it
        if theButton is "Arm" then
            set armed of eachCue to true
        else if theButton is "Disarm" then
            set armed of eachCue to false
        else
            set armed of eachCue to not armed of eachCue
        end if
    end repeat
end tell
```

Clear Cue Number

Clear Cue Number of selected cue(s) that aren't directly in a cue list (the script will also optionally act on children of selected Group Cues):

```
set userEnterIntoGroups to true -- Set this to false if you don't want the
script to act on children of selected Group Cues

tell front workspace
    set cuesToProcess to (selected as list)
    set selectedCount to count cuesToProcess
    set currentCueList to current cue list
    set i to 0
    repeat until i = (count cuesToProcess)
        set eachCue to item (i + 1) of cuesToProcess
        if i < selectedCount then -- Don't need to check parentage of cues
added to the list as children of selected Group Cues
            if parent of eachCue is not currentCueList then
                set q number of eachCue to ""
            end if
        else
            set q number of eachCue to ""
        end if
        if userEnterIntoGroups is true then
```

```
            if q type of eachCue is "Group" then
                set cuesToProcess to cuesToProcess & cues of eachCue
            end if
        end if
        set i to i + 1
    end repeat
end tell
```

Renumber with a prefix

Renumber selected cue(s) using a sequence list, starting with the number you enter and incrementing the (integer) numerical part at the end; QLab's logic of skipping existing numbers is modelled:

```
set userDefaultIncrement to 1 -- Use this to specify the default increment
between numbers presented in the dialog

-- Declarations

global dialogTitle
set dialogTitle to "Renumber with a prefix"

-- Check the Clipboard for a previous prefix

try
    set clipboardContents to the clipboard as text
on error
    set clipboardContents to ""
end try

if clipboardContents contains return or clipboardContents contains linefeed
then -- Slight protection against spurious Clipboard contents
    set clipboardContents to ""
end if

-- Main routine

set startingNumber to enterSomeTextWithIcon("Enter the Cue Number for the
first selected cue:", clipboardContents, true)

set thePrefix to startingNumber
set theSuffix to ""
set nonNumberFound to false

repeat with i from (count characters of startingNumber) to 1 by -1
    if character i of startingNumber is not in characters of "0123456789"
then
        set nonNumberFound to true
        set thePrefix to (characters 1 thru i of startingNumber) as text
        try -- If the last character is not a number then theSuffix remains
as ""
```

```
            set theSuffix to (characters (i + 1) thru end of startingNumber)
as text
        end try
        exit repeat
    end if
end repeat

if nonNumberFound is false then -- Edge case where the text entered is a
number with no prefix
    set thePrefix to ""
    set theSuffix to startingNumber
end if

set theSuffix to theSuffix as integer

set theIncrement to enterANumberWithIcon("Enter the increment:",
userDefaultIncrement)

tell front workspace

    set selectedCues to (selected as list)

    -- Clear existing Cue Numbers

    repeat with eachCue in selectedCues
        set q number of eachCue to ""
    end repeat

    -- Get a list of numbers that can't be used

    set allNumbers to q number of cues
    set allNumbersRef to a reference to allNumbers

    -- Renumber the cues

    repeat with eachCue in selectedCues
        set newNumber to (thePrefix & theSuffix) as text
        repeat until newNumber is not in allNumbersRef -- If the number is
in use, then skip it
            set theSuffix to theSuffix + theIncrement
            set newNumber to (thePrefix & theSuffix) as text
        end repeat
        set q number of eachCue to newNumber
        set theSuffix to theSuffix + theIncrement
    end repeat

end tell

-- Copy the prefix to the Clipboard

set the clipboard to startingNumber as text
```

```
-- Subroutines

(* === INPUT === *)

on enterANumberWithIcon(thePrompt, defaultAnswer) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set theQuestion to text returned of (display dialog thePrompt
with title dialogTitle with icon 1 ¬
                default answer defaultAnswer buttons {"Cancel", "OK"}
default button "OK" cancel button "Cancel")
            try
                set theAnswer to theQuestion as number
            on error
                set theQuestion to ""
            end try
        end repeat
        return theAnswer
    end tell
end enterANumberWithIcon

on enterSomeTextWithIcon(thePrompt, defaultAnswer, emptyAllowed) -- [Shared
subroutine]
    tell application id "com.figure53.QLab.4"
        set theAnswer to ""
        repeat until theAnswer is not ""
            set theAnswer to text returned of (display dialog thePrompt with
title dialogTitle with icon 1 ¬
                default answer defaultAnswer buttons {"Cancel", "OK"}
default button "OK" cancel button "Cancel")
            if emptyAllowed is true then exit repeat
        end repeat
        return theAnswer
    end tell
end enterSomeTextWithIcon
```

Add suffix to renumber

Append the string you enter to the Cue Numbers of the selected cue(s):

```
-- Declarations

global dialogTitle
set dialogTitle to "Add suffix to Cue Number"

-- Check the Clipboard for a previous suffix

try
```

```
    set clipboardContents to the clipboard as text
on error
    set clipboardContents to ""
end try

if clipboardContents contains return or clipboardContents contains linefeed
then -- Slight protection against spurious Clipboard contents
    set clipboardContents to ""
end if

-- Main routine

set theSuffix to enterSomeTextWithIcon("Enter the suffix to append:",
clipboardContents, true)

tell front workspace
    repeat with eachCue in (selected as list)
        set q number of eachCue to (q number of eachCue & theSuffix)
    end repeat
end tell

-- Copy the suffix to the Clipboard

set the clipboard to theSuffix as text

-- Subroutines

(* === INPUT === *)

on enterSomeTextWithIcon(thePrompt, defaultAnswer, emptyAllowed) -- [Shared
subroutine]
    tell application id "com.figure53.QLab.4"
        set theAnswer to ""
        repeat until theAnswer is not ""
            set theAnswer to text returned of (display dialog thePrompt with
title dialogTitle with icon 1 ¬
                default answer defaultAnswer buttons {"Cancel", "OK"}
default button "OK" cancel button "Cancel")
            if emptyAllowed is true then exit repeat
        end repeat
        return theAnswer
    end tell
end enterSomeTextWithIcon
```

Reset names

Reset name of selected cue(s) (the script will reset the name of an Audio, Video or MIDI File Cue to its File Target, name other types of cue based on their type and Cue Target, name MIDI Cues based on their parameters, or reset Network Cues to their default names; other cues that have neither File nor Cue Target are unaffected):

```
-- This script can optionally convert note numbers in MIDI Cues to note
names and mark looped cues: use the variables below to set the behaviour

set userMIDIConversionMode to item 1 of {"Numbers", "Names", "Both"} -- Use
this to choose how MIDI note values will be included in cue names
set userNote60Is to item 1 of {"C3", "C4"} -- Use this to set which note
name corresponds to note 60

set userFlagInfiniteLoops to true -- Set whether to append the comments
below to the default names of Audio & Video Cues that contain infinite loops
set userInfiniteCueFlag to "!! Infinite loop !!"
set userInfiniteSliceFlag to "!! Infinite slice !!"

-- Declarations

global userMIDIConversionMode, noteNames, octaveConvention
set noteNames to {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A",
"A#", "B"}
if userNote60Is is "C3" then
    set octaveConvention to -2
else
    set octaveConvention to -1
end if

set mscCommandFormat to {1, "Lighting (General)", 2, "Moving Lights", 3,
"Color Changers", 4, "Strobes", 5, "Lasers", 6, "Chasers", ¬
    16, "Sound (General)", 17, "Music", 18, "CD Players", 19, "EPROM
Playback", 20, "Audio Tape Machines", 21, "Intercoms", 22, "Amplifiers", ¬
    23, "Audio Effects Devices", 24, "Equalizers", 32, "Machinery
(General)", 33, "Rigging", 34, "Flys", 35, "Lifts", 36, "Turntables", 37,
"Trusses", ¬
    38, "Robots", 39, "Animation", 40, "Floats", 41, "Breakaways", 42,
"Barges", 48, "Video (General)", 49, "Video Tape Machines", ¬
    50, "Video Cassette Machines", 51, "Video Disc Players", 52, "Video
Switchers", 53, "Video Effects", 54, "Video Character Generators", ¬
    55, "Video Still Stores", 56, "Video Monitors", 64, "Projection
(General)", 65, "Film Projectors", 66, "Slide Projectors", 67, "Video
Projectors", ¬
    68, "Dissolvers", 69, "Shutter Controls", 80, "Process Control
(General)", 81, "Hydraulic Oil", 82, "H2O", 83, "CO2", 84, "Compressed Air",
¬
    85, "Natural Gas", 86, "Fog", 87, "Smoke", 88, "Cracked Haze", 96,
"Pyrotechnics (General)", 97, "Fireworks", 98, "Explosions", 99, "Flame", ¬
    100, "Smoke Pots", 127, "All Types"}
set mscCommandNumber to {1, "GO", 2, "STOP", 3, "RESUME", 4, "TIMED_GO", 5,
"LOAD", 6, "SET", 7, "FIRE", 8, "ALL_OFF", ¬
    9, "RESTORE", 10, "RESET", 11, "GO_OFF", 16, "GO/JAM_CLOCK", 17,
"STANDBY_+", 18, "STANDBY_-", 19, "SEQUENCE_+", ¬
    20, "SEQUENCE_-", 21, "START_CLOCK", 22, "STOP_CLOCK", 23, "ZERO_CLOCK",
24, "SET_CLOCK", 25, "MTC_CHASE_ON", ¬
    26, "MTC_CHASE_OFF", 27, "OPEN_CUE_LIST", 28, "CLOSE_CUE_LIST", 29,
```

```
"OPEN_CUE_PATH", 30, "CLOSE_CUE_PATH"}
set mscCommandTakesQNumber to {1, 2, 3, 4, 5, 11, 16}
set mscCommandTakesQList to {1, 2, 3, 4, 5, 11, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28}
set mscCommandTakesQPath to {1, 2, 3, 4, 5, 11, 16, 29, 30}
set mscCommandTakesMacro to {7}
set mscCommandTakesControl to {6}
set mscCommandTakesTC to {4, 6, 24}

-- Main routine

tell front workspace
    repeat with eachCue in (selected as list)
        try -- Audio, Video or MIDI File Cues
            set eachFile to file target of eachCue as alias
            tell application "System Events"
                set eachNameList to {name of eachFile}
            end tell
            try
                if infinite loop of eachCue is true then
                    set end of eachNameList to userInfiniteCueFlag
                end if
                if last slice infinite loop of eachCue is true then
                    set end of eachNameList to userInfiniteSliceFlag
                end if
            end try
            set currentTIDs to AppleScript's text item delimiters
            set AppleScript's text item delimiters to " | "
            set q name of eachCue to eachNameList as text
            set AppleScript's text item delimiters to currentTIDs
        on error
            try -- Fade, Start, Stop, Pause, Load, Reset, Devamp, GoTo,
Target, Arm or Disarm Cues
                set eachTarget to q list name of cue target of eachCue
                if eachTarget is "" then
                    set eachTarget to q display name of cue target of
eachCue
                end if
                set eachType to q type of eachCue
                if eachType is "Fade" then
                    if stop target when done of eachCue is true then
                        set q name of eachCue to "Fade out: " & eachTarget
                    else
                        set q name of eachCue to "Fade: " & eachTarget
                    end if
                else
                    set q name of eachCue to eachType & ": " & eachTarget
                end if
            on error
                try -- MIDI Cues
                    set messageType to message type of eachCue
```

```
                    if messageType is voice then
                        set eachChannel to channel of eachCue
                        set eachCommand to command of eachCue
                        if eachCommand is note_on then
                            set byteOne to byte one of eachCue
                            set byteTwo to byte two of eachCue
                            set q name of eachCue to "Channel " &
eachChannel & " | Note On | " & my convertMIDINoteValues(byteOne, byteTwo)
                        else if eachCommand is note_off then
                            set byteOne to byte one of eachCue
                            set byteTwo to byte two of eachCue
                            set q name of eachCue to "Channel " &
eachChannel & " | Note Off | " & my convertMIDINoteValues(byteOne, byteTwo)
                        else if eachCommand is program_change then
                            set byteOne to byte one of eachCue
                            set q name of eachCue to "Channel " &
eachChannel & " | Program Change | " & byteOne
                        else if eachCommand is control_change then
                            set byteOne to byte one of eachCue
                            set byteTwo to byte two of eachCue
                            if integrated fade of eachCue is disabled then
                                set q name of eachCue to "Channel " &
eachChannel & " | Control Change | " & byteOne & " @ " & byteTwo
                            else
                                set endValue to end value of eachCue
                                set q name of eachCue to "Channel " &
eachChannel & " | Control Change | " & byteOne & " @ " & byteTwo ¬
                                    & " … " & endValue
                            end if
                        else if eachCommand is key_pressure then
                            set byteOne to byte one of eachCue
                            set byteTwo to byte two of eachCue
                            if integrated fade of eachCue is disabled then
                                set q name of eachCue to "Channel " &
eachChannel & " | Key Pressure | " & my convertMIDINoteValues(byteOne,
byteTwo)
                            else
                                set endValue to end value of eachCue
                                set q name of eachCue to ¬
                                    "Channel " & eachChannel & " | Key
Pressure | " & my convertMIDINoteValues(byteOne, byteTwo & " … " & endValue)
                            end if
                        else if eachCommand is channel_pressure then
                            set byteOne to byte one of eachCue
                            set q name of eachCue to "Channel " &
eachChannel & " | Channel Pressure | " & byteOne
                            if integrated fade of eachCue is disabled then
                            else
                                set endValue to end value of eachCue
                                set q name of eachCue to "Channel " &
eachChannel & " | Channel Pressure | " & byteOne & " … " & endValue
```

```
                            end if
                        else if eachCommand is pitch_bend then
                            set byteCombo to (byte combo of eachCue) - 8192
                            if integrated fade of eachCue is disabled then
                                set q name of eachCue to "Channel " &
eachChannel & " | Pitch Bend | " & byteCombo
                            else
                                set endValue to (end value of eachCue) -
8192
                                set q name of eachCue to "Channel " &
eachChannel & " | Pitch Bend | " & byteCombo & " … " & endValue
                            end if
                        end if
                    else if messageType is msc then
                        set mscProperties to {}
                        set mscEmptyQNumber to false
                        set mscEmptyQList to false
                        set end of mscProperties to my lookUp(command format
of eachCue, mscCommandFormat)
                        set end of mscProperties to {"Device ID " & deviceID
of eachCue}
                        set mscCommand to command number of eachCue
                        set end of mscProperties to my lookUp(mscCommand,
mscCommandNumber)
                        if mscCommand is in mscCommandTakesQNumber then
                            set mscQNumber to q_number of eachCue
                            if mscQNumber is not "" then
                                set end of mscProperties to "Q " &
mscQNumber
                            else
                                set mscEmptyQNumber to true
                            end if
                        end if
                        if mscCommand is in mscCommandTakesQList and
mscEmptyQNumber is false then
                            set mscQList to q_list of eachCue
                            if mscQList is not "" then
                                set end of mscProperties to "List " &
mscQList
                            else
                                set mscEmptyQList to true
                            end if
                        end if
                        if mscCommand is in mscCommandTakesQPath and
mscEmptyQNumber is false and mscEmptyQList is false then
                            set mscQPath to q_path of eachCue
                            if mscQPath is not "" then set end of
mscProperties to "Path " & mscQPath
                        end if
                        if mscCommand is in mscCommandTakesMacro then
                            set end of mscProperties to macro of eachCue
```

```
                        end if
                        if mscCommand is in mscCommandTakesControl then
                            set end of mscProperties to (control number of
eachCue & " @ " & control value of eachCue) as text
                        end if
                        if mscCommand is in mscCommandTakesTC and send time
with set of eachCue is true then
                            set end of mscProperties to my padNumber(hours
of eachCue, 2) & ¬
                                ":" & my padNumber(minutes of eachCue, 2) &
¬
                                ":" & my padNumber(seconds of eachCue, 2) &
¬
                                ":" & my padNumber(frames of eachCue, 2) & ¬
                                "/" & my padNumber(subframes of eachCue, 2)
-- Not sophisticated enough to use ";" for drop frame rates
                        end if
                        set currentTIDs to AppleScript's text item
delimiters
                        set AppleScript's text item delimiters to " | "
                        set q name of eachCue to mscProperties as text
                        set AppleScript's text item delimiters to
currentTIDs
                    else
                        set q name of eachCue to "MIDI SysEx"
                    end if
                on error -- Network Cues
                    if q type of eachCue is "Network" then
                        set q name of eachCue to "" -- Reset to default
                    end if
                end try
            end try
        end try
    end repeat
end tell

-- Subroutines

(* === DATA WRANGLING === *)

on lookUp(lookUpValue, lookUpTable) -- [Shared subroutine]
    repeat with i from 1 to (count lookUpTable) by 2
        if lookUpValue is item i of lookUpTable then
            return item (i + 1) of lookUpTable
        end if
    end repeat
end lookUp

(* === TEXT WRANGLING === *)

on configureNoteNameString(noteNumber) -- [Shared subroutine]
```

```
    set theOctave to (noteNumber div 12) + octaveConvention
    set theNote to item (noteNumber mod 12 + 1) of noteNames
    set noteNameString to theNote & theOctave
    return noteNameString
end configureNoteNameString

on convertMIDINoteValues(noteNumber, noteVelocity) -- [Shared subroutine]
    if userMIDIConversionMode is "Numbers" then
        return noteNumber & " @ " & noteVelocity
    else if userMIDIConversionMode is "Names" then
        return my configureNoteNameString(noteNumber) & " @ " & noteVelocity
    else if userMIDIConversionMode is "Both" then
        return noteNumber & " @ " & noteVelocity & " | " & my
configureNoteNameString(noteNumber)
    end if
end convertMIDINoteValues

on padNumber(theNumber, minimumDigits) -- [Shared subroutine]
    set paddedNumber to theNumber as text
    repeat while (count paddedNumber) < minimumDigits
        set paddedNumber to "0" & paddedNumber
    end repeat
    return paddedNumber
end padNumber
```

Update filenames

Update filename of File Target for selected cue(s) (the script will change the name of the File Target on disk for an Audio, Video or MIDI File Cue to be the same as the custom name of the cue; QLab's display won't update until you change the selection):

```
tell front workspace
    repeat with eachCue in (selected as list)
        try
            set eachName to q name of eachCue
            if eachName is "" then -- Skip blank names as otherwise you make
invisible files!
                error
            end if
            set eachTarget to file target of eachCue as alias
            tell application "System Events"
                set eachExtension to name extension of eachTarget
                if eachName does not end with eachExtension then
                    set name of eachTarget to eachName & "." & eachExtension
                else
                    set name of eachTarget to eachName
                end if
            end tell
        end try
    end repeat
```

```
end tell
```

Clear Notes

Clear Notes of selected cue(s):

```
/cue/selected/notes ""
```

Move cut cues

Move selected cue(s) to “Cut” cue list:

```
set userCutList to "Cut" -- Use this to set the name of the cue list to
which to move cut cues

-- Declarations

global dialogTitle
set dialogTitle to "Move cut cues"

-- Main routine

tell front workspace

    try
        set cutList to first cue list whose q name is userCutList
    on error
        display dialog "The destination cue list can not be found…" with
title dialogTitle with icon 0 buttons {"OK"} default button "OK" giving up
after 5
        return
    end try
    if current cue list is not cutList then
        set selectedCues to (selected as list)
        set selectedCueIDs to {}
        repeat with eachCue in selectedCues
            set end of selectedCueIDs to uniqueID of eachCue
        end repeat
        repeat with eachCueID in selectedCueIDs
            set eachParentID to uniqueID of parent of cue id eachCueID
            if eachParentID is not in selectedCueIDs then -- If the parent
cue is being cut too, keep the nested structure
                move cue id eachCueID of cue id eachParentID to end of
cutList
            end if
        end repeat
    end if
end tell
```

Delete

Delete selected cue(s) including their File Targets (target files will be moved to the Trash):

```
tell front workspace
    repeat with eachCue in (selected as list)
        try
            set eachTarget to file target of eachCue
            if eachTarget as text does not contain ".Trash" then
                tell application "Finder"
                    delete eachTarget
                end tell
            end if
        end try
        set eachCueID to uniqueID of eachCue
        delete cue id eachCueID of parent of eachCue
    end repeat
end tell
```

Batch adjust selected

This script will attempt to batch adjust properties of the currently selected cues according to your instructions... (it won't fire unless you have a cue selected, but once running you can choose to act on all cues):

```
-- ###FIXME### Impact of housing such a large script in the workspace
unknown!
-- ###FIXME### Testing has not been _exhaustive_, for obvious reasons!
-- ###FIXME### Unclear whether script can run in the background while you
continue working in QLab (or elsewhere)
-- ###TODO### Add Gangs, MSC command formats?
-- ###TODO### Include "considering case" option for searches

-- Best run as a separate process so it can be happening in the background

set userEscapeHatchInterval to 50 -- Set the number of cues to process
between each progress report / opportunity to cancel

-- Explanations

set theExplanation to "This script will attempt to batch adjust properties
of the currently selected cues according to your instructions.

There is some error protection, but it is impossible to make this process
completely idiot-proof. You should be warned if something threw an error, "
& ¬
    "but it's not possible to track exactly what it was."

-- Declarations
```

```
global dialogTitle, qLabMaxAudioInputs, qLabMaxAudioChannels,
userEscapeHatchInterval, startTime, ohDear, abortAbort
set dialogTitle to "Batch adjust selected"

set qLabMaxAudioInputs to 24
set qLabMaxAudioChannels to 64

global subChoiceTimesParameter, subChoiceMIDIParameter,
subChoiceMIDICommand, subChoiceAutoload, subChoiceArmed,
subChoiceContinueMode, subChoiceMode
(* These lists are also used for lookup within the chosen handlers *)

set processChoices to {"Levels", "File Target", "File Target (keeping
times)", "Name", "Number", "Notes", "Times", "MIDI", "MSC Device ID", ¬
    "Auto-load", "Armed", "Continue Mode", "Mode", "Patch", "Camera Patch",
"Command Text", "…Finished adjusting", "** Set flag to process ALL CUES!
**"}
(* Although batch adjusting of Continue Mode is now built-in, it only
operates on selected cues — so keeping it for now as this script can affect
ALL cues *)

set subChoiceName to {"Set/reset", "Basic search & replace", "Add prefix",
"Add suffix", "Make series"}
set subChoiceNumber to {"Add prefix", "Add suffix", "Make series"}
set subChoiceNotes to {"Clear", "Set", "Basic search & replace", "Add
prefix", "Add suffix"}
set subChoiceTimesParameter to {"Pre Wait", "Duration", "Post Wait"} --
These values can be customised as they are never used explicitly
set subChoiceTimes to {"Set", "Scale", "Add/subtract amount"}
set subChoiceMIDIParameter to {"Command", "Channel", "Byte One", "Byte Two",
"Byte Combo", ¬
    "End Value"} -- These values can be customised as they are never used
explicitly
set subChoiceMIDICommand to {"Note On", "Note Off", "Program Change",
"Control Change", ¬
    "Key Pressure", "Channel Pressure", "Pitch Bend"} -- These values can be
customised as they are never used explicitly
set subChoiceMIDI to {"Set", "Scale", "Add/subtract amount", "Make series"}
set subChoiceAutoload to {"On", "Off"} -- These values can be customised as
they are never used explicitly
set subChoiceArmed to {"Armed", "Disarmed"} -- These values can be
customised as they are never used explicitly
set subChoiceContinueMode to {"Do not continue", "Auto-continue", "Auto-
follow"} -- These values can be customised as they are never used explicitly
set subChoiceMode to {"Timeline - Start all children simultaneously", "Start
first child and enter into group", "Start first child and go to next cue", ¬
    "Start random child and go to next cue"} -- These values can be
customised as they are never used explicitly
set subChoiceCameraPatch to {1, 2, 3, 4, 5, 6, 7, 8}
set subChoiceCommandText to {"Basic search & replace (all text)", ¬
```

```
    "Basic search & replace (Instrument Names only)"} -- These values can be
customised as they are never used explicitly

-- NB: use "false " rather than "false" in lists presented to pickFromList()

-- Preamble

set theProcess to ""
set firstTime to true
repeat until theProcess is "…Finished adjusting"

    set ohDear to false -- A simple flag to detect problems
    set abortAbort to false -- A flag for aborting!

    -- Test for a selection; modify options if only one cue selected

    if firstTime is true then -- Only need to do this step once

        tell application id "com.figure53.QLab.4"
            set theSelection to (selected of front workspace as list)
        end tell

        set theSelectionRef to a reference to theSelection
        set countSelection to count theSelectionRef
        if countSelection is 0 then
            return
        end if

        if countSelection is 1 then
            set subChoiceName to items 1 through -2 of subChoiceName --
Remove "Make series"
            set subChoiceNumber to items 1 through -2 of subChoiceNumber --
Remove "Make series"
            set subChoiceMIDI to items 1 through -2 of subChoiceMIDI --
Remove "Make series"
        end if

    end if

    -- Choose a process

    set theProcess to pickFromList(processChoices, theExplanation & return &
return & ¬
        "So that you can run more than one process, you'll keep coming back
to this screen until you hit any \"Cancel\" button, " & ¬
        "or choose \"…Finished adjusting\"." & return & return & "Choose a
property category:")

    -- Deal with "process ALL CUES" flag

    if firstTime is true then -- Only need to do this step once
```

```
        set processChoices to items 1 through -2 of processChoices -- Remove
"** Set flag to process ALL CUES! **"

        if theProcess is "** Set flag to process ALL CUES! **" then

            tell application id "com.figure53.QLab.4"
                set theSelection to (cues of front workspace as list)
            end tell

            set theExplanation to "WARNING: acting on ALL CUES!" & return &
"WARNING: acting on ALL CUES!" & return & "WARNING: acting on ALL CUES!"

        end if

        set firstTime to false

    end if

    -- Find out more about what we're doing, and then call a subroutine to
do it…

    if theProcess is "Levels" then

        adjustLevels(theSelectionRef, "selected cues")

    else if theProcess is "File Target" then

        adjustFileTarget(theSelectionRef, "Set", "selected cues")

    else if theProcess is "File Target (keeping times)" then

        adjustFileTarget(theSelectionRef, "Change", "selected cues")

    else if theProcess is "Name" then

        set theChoice to pickFromList(subChoiceName, "Choose how you would
like to adjust the names of the selected cues:")

        if theChoice is "Set/reset" then
            adjustSetName(theSelectionRef, "selected cues")
        else if theChoice is "Basic search & replace" then
            adjustSearchReplaceName(theSelectionRef, "selected cues")
        else if theChoice is "Add prefix" then
            adjustPrefixName(theSelectionRef, "selected cues")
        else if theChoice is "Add suffix" then
            adjustSuffixName(theSelectionRef, "selected cues")
        else if theChoice is "Make series" then
            adjustSeriesName(theSelectionRef, "selected cues")
        end if
```

```
    else if theProcess is "Number" then

        set theChoice to pickFromList(subChoiceNumber, "Choose how you would
like to adjust the Cue Numbers of the selected cues:")

        if theChoice is "Add prefix" then
            adjustPrefixNumber(theSelectionRef, "selected cues")
        else if theChoice is "Add suffix" then
            adjustSuffixNumber(theSelectionRef, "selected cues")
        else if theChoice is "Make series" then
            adjustSeriesNumber(theSelectionRef, "selected cues")
        end if

    else if theProcess is "Notes" then

        set theChoice to pickFromList(subChoiceNotes, "Choose how you would
like to adjust the Notes of the selected cues " & ¬
            "(NB: scripting of Notes is plain-text only):")

        if theChoice is "Clear" then
            adjustClearNotes(theSelectionRef, "selected cues")
        else if theChoice is "Set" then
            adjustSetNotes(theSelectionRef, "selected cues")
        else if theChoice is "Basic search & replace" then
            adjustSearchReplaceNotes(theSelectionRef, "selected cues")
        else if theChoice is "Add prefix" then
            adjustPrefixNotes(theSelectionRef, "selected cues")
        else if theChoice is "Add suffix" then
            adjustSuffixNotes(theSelectionRef, "selected cues")
        end if

    else if theProcess is "Times" then

        set parameterChoice to pickFromList(subChoiceTimesParameter, "Choose
the time parameter to adjust:")
        set theChoice to pickFromList(subChoiceTimes, "Choose how you would
like to adjust the " & parameterChoice & " of the selected cues:")

        if theChoice is "Set" then
            adjustSetTime(theSelectionRef, parameterChoice, "selected cues")
        else if theChoice is "Scale" then
            adjustScaleTime(theSelectionRef, parameterChoice, "selected
cues")
        else if theChoice is "Add/subtract amount" then
            adjustAddSubractTime(theSelectionRef, parameterChoice, "selected
cues")
        end if

    else if theProcess is "MIDI" then

        -- subChoiceMIDIParameter = {"Command", "Channel", "Byte One", "Byte
```

```
Two", "Byte Combo", "End Value"}

        set parameterChoice to pickFromList(subChoiceMIDIParameter, ¬
            "Choose the MIDI parameter to adjust (\"" & item 5 of
subChoiceMIDIParameter & "\" will only affect pitch bend commands):")
        if parameterChoice is item 1 of subChoiceMIDIParameter then
            set theChoice to item 1 of subChoiceMIDIParameter
        else if parameterChoice is item 2 of subChoiceMIDIParameter then
            set theChoice to "Set" -- The other options don't make a lot of
sense for channel!
        else
            set theChoice to pickFromList(subChoiceMIDI, "Choose how you
would like to adjust the " & parameterChoice & " of the selected cues:")
        end if

        if theChoice is item 1 of subChoiceMIDIParameter then
            adjustSetMIDICommand(theSelectionRef, "selected cues")
        else if theChoice is "Set" then
            adjustSetMIDI(theSelectionRef, parameterChoice, "selected cues")
        else if theChoice is "Scale" then
            adjustScaleMIDI(theSelectionRef, parameterChoice, "selected
cues")
        else if theChoice is "Add/subtract amount" then
            adjustAddSubractMIDI(theSelectionRef, parameterChoice, "selected
cues")
        else if theChoice is "Make series" then
            adjustSeriesMIDI(theSelectionRef, parameterChoice, "selected
cues")
        end if

    else if theProcess is "MSC Device ID" then

        adjustDeviceID(theSelectionRef, "selected cues")

    else if theProcess is "Auto-load" then

        set parameterChoice to pickFromList(subChoiceAutoload, "Set Auto-
load of the selected cues to:")

        adjustAutoload(theSelectionRef, parameterChoice, "selected cues")

    else if theProcess is "Armed" then

        set parameterChoice to pickFromList(subChoiceArmed, "Set Armed of
the selected cues to:")

        adjustArmed(theSelectionRef, parameterChoice, "selected cues")

    else if theProcess is "Continue Mode" then

        set parameterChoice to pickFromList(subChoiceContinueMode, "Set the
```

```
Continue Mode of the selected cues to:")

        adjustContinueMode(theSelectionRef, parameterChoice, "selected
cues")

    else if theProcess is "Mode" then

        set parameterChoice to pickFromList(subChoiceMode, "Set the Mode of
the selected cues to:")

        adjustMode(theSelectionRef, parameterChoice, "selected cues")

    else if theProcess is "Patch" then

        adjustPatch(theSelectionRef, "selected cues")

    else if theProcess is "Camera Patch" then

        set parameterChoice to pickFromList(subChoiceCameraPatch, "Set the
Camera Patch of the selected cues to:")

        adjustCameraPatch(theSelectionRef, parameterChoice, "selected cues")

    else if theProcess is "Command Text" then

        set theChoice to pickFromList(subChoiceCommandText, "Choose how you
would like to adjust the Command Text of the selected cues:")

        if theChoice is "Basic search & replace (all text)" then
            adjustSearchReplaceCommandText(theSelectionRef, "selected cues",
false)
        else if theChoice is "Basic search & replace (Instrument Names
only)" then
            adjustSearchReplaceCommandText(theSelectionRef, "selected cues",
true)
        end if

    end if

end repeat

-- Subroutines

(* === INPUT === *)

on enterAnInteger(thePrompt, defaultAnswer) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set theQuestion to text returned of (display dialog thePrompt
with title dialogTitle default answer defaultAnswer buttons {"Cancel", "OK"}
```

```
¬
                default button "OK" cancel button "Cancel")
            try
                set theAnswer to theQuestion as integer
            on error
                set theQuestion to ""
            end try
        end repeat
        return theAnswer
    end tell
end enterAnInteger

on enterANumber(thePrompt, defaultAnswer) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set theQuestion to text returned of (display dialog thePrompt
with title dialogTitle default answer defaultAnswer buttons {"Cancel", "OK"}
¬
                default button "OK" cancel button "Cancel")
            try
                set theAnswer to theQuestion as number
            on error
                set theQuestion to ""
            end try
        end repeat
        return theAnswer
    end tell
end enterANumber

on enterANumberWithRangeWithCustomButton(thePrompt, defaultAnswer, ¬
    lowRange, acceptEqualsLowRange, highRange, acceptEqualsHighRange,
integerOnly, customButton, defaultButton) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
of (display dialog thePrompt with title dialogTitle ¬
                default answer defaultAnswer buttons (customButton as list)
& {"Cancel", "OK"} default button defaultButton cancel button "Cancel")
            if theButton is customButton then
                set theAnswer to theButton
                exit repeat
            end if
            try
                if integerOnly is true then
                    set theAnswer to theQuestion as integer -- Detects non-
numeric strings
                    if theAnswer as text is not theQuestion then -- Detects
non-integer input
                        set theQuestion to ""
```

```
                    end if
                else
                    set theAnswer to theQuestion as number -- Detects non-
numeric strings
                end if
                if lowRange is not false then
                    if acceptEqualsLowRange is true then
                        if theAnswer < lowRange then
                            set theQuestion to ""
                        end if
                    else
                        if theAnswer ≤ lowRange then
                            set theQuestion to ""
                        end if
                    end if
                end if
                if highRange is not false then
                    if acceptEqualsHighRange is true then
                        if theAnswer > highRange then
                            set theQuestion to ""
                        end if
                    else
                        if theAnswer ≥ highRange then
                            set theQuestion to ""
                        end if
                    end if
                end if
            on error
                set theQuestion to ""
            end try
        end repeat
        return theAnswer
    end tell
end enterANumberWithRangeWithCustomButton

on enterARatio(thePrompt, defaultAnswer) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set theQuestion to text returned of (display dialog thePrompt
with title dialogTitle default answer defaultAnswer buttons {"Cancel", "OK"}
¬
                default button "OK" cancel button "Cancel")
            try
                set theAnswer to theQuestion as number
                if theAnswer ≤ 0 then
                    set theQuestion to ""
                end if
            on error
                set theQuestion to ""
            end try
```

```
        end repeat
        return theAnswer
    end tell
end enterARatio

on enterATimeWithCustomButton(thePrompt, defaultAnswer, customButton) --
[Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
of (display dialog thePrompt with title dialogTitle ¬
                default answer defaultAnswer buttons (customButton as list)
& {"Cancel", "OK"} default button "OK" cancel button "Cancel")
            if theButton is customButton then
                set theAnswer to theButton
                exit repeat
            end if
            try
                set theAnswer to theQuestion as number
                if theAnswer < 0 then
                    set theQuestion to ""
                end if
            on error
                if theQuestion contains ":" then
                    set theAnswer to my makeSecondsFromM_S(theQuestion)
                    if theAnswer is false then
                        set theQuestion to ""
                    end if
                else
                    set theQuestion to ""
                end if
            end try
        end repeat
        return theAnswer
    end tell
end enterATimeWithCustomButton

on enterSomeText(thePrompt, defaultAnswer, emptyAllowed) -- [Shared
subroutine]
    tell application id "com.figure53.QLab.4"
        set theAnswer to ""
        repeat until theAnswer is not ""
            set theAnswer to text returned of (display dialog thePrompt with
title dialogTitle default answer defaultAnswer buttons {"Cancel", "OK"} ¬
                default button "OK" cancel button "Cancel")
            if emptyAllowed is true then exit repeat
        end repeat
        return theAnswer
    end tell
end enterSomeText
```

```
on pickFromList(theChoice, thePrompt) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        choose from list theChoice with prompt thePrompt with title
dialogTitle default items item 1 of theChoice
        if result is not false then
            return item 1 of result
        else
            error number -128
        end if
    end tell
end pickFromList

(* === OUTPUT === *)

on startTheClock() -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        display dialog "One moment caller…" with title dialogTitle with icon
1 buttons {"OK"} default button "OK" giving up after 1
    end tell
    set startTime to current date
end startTheClock

on countdownTimer(thisStep, totalSteps, whichCuesString) -- [Shared
subroutine]
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeMSS(timeTaken)
    tell application id "com.figure53.QLab.4"
        if frontmost then
            display dialog "Time elapsed: " & timeString & " — " & thisStep
& " of " & totalSteps & " " & whichCuesString & ¬
                " done…" with title dialogTitle with icon 1 buttons
{"Cancel", "OK"} default button "OK" cancel button "Cancel" giving up after
1
        end if
    end tell
end countdownTimer

on finishedDialogBespoke()
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeNiceT(timeTaken)
    tell application id "com.figure53.QLab.4"
        activate
        if abortAbort is true then
            display dialog "Process aborted due to errors!" with title
dialogTitle with icon 0 buttons {"OK"} default button "OK" giving up after
120
        else
            if ohDear is true then
```

```
                set ohDearString to " There were some errors, so you should
check the results."
                set ohDearIcon to 0
            else
                set ohDearString to ""
                set ohDearIcon to 1
            end if
            display dialog "Done." & ohDearString & return & return & "(That
took " & timeString & ".)" with title dialogTitle with icon ohDearIcon ¬
                buttons {"OK"} default button "OK" giving up after 60
        end if
    end tell
end finishedDialogBespoke

(* === PROCESSING === *)

on adjustLevels(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        -- Get the levels

        set validEntry to false
        set previousTry to ""
        repeat until validEntry is true

            set levelsString to my enterSomeText("Enter the levels you wish
to adjust as
\"row/column/delta\" or \"row/column/@level\"; you can separate multiple
entries with spaces.

For example, \"0/0/-10 0/2/@-20\" will take 10dB from the Master level (row
0 column 0) and set the Output 2 level (row 0 column 2) to -20dB.", ¬
                previousTry, false)
            set previousTry to levelsString

            -- Convert string to array

            set currentTIDs to AppleScript's text item delimiters
            set AppleScript's text item delimiters to space
            set levelsWords to text items of levelsString
            set howManyLevels to count levelsWords
            set AppleScript's text item delimiters to "/"
            set backToText to levelsWords as text
            set levelsArray to text items of backToText
            set countLevelsArray to count levelsArray
            set AppleScript's text item delimiters to currentTIDs

            -- Check for validity

            if howManyLevels * 3 is countLevelsArray then -- First hurdle
```

```
                set validEntry to true
                try
                    repeat with i from 1 to countLevelsArray by 3
                        set eachRow to (item i of levelsArray) as number
                        set eachColumn to (item (i + 1) of levelsArray) as
number
                        set eachLevel to item (i + 2) of levelsArray
                        if eachRow < 0 or eachRow > qLabMaxAudioInputs then
-- Check for valid row
                            set validEntry to false
                            exit repeat
                        end if
                        if eachColumn < 0 or eachColumn >
qLabMaxAudioChannels then -- Check for valid column
                            set validEntry to false
                            exit repeat
                        end if
                        if eachLevel does not start with "@" then -- Check
for valid level
                            if (eachLevel as number) is 0 then -- Delta
level can't be 0 (also checks string is a number)
                                set validEntry to false
                                exit repeat
                            end if
                        else
                            set finalCheck to (rest of characters of
eachLevel as text) as number -- Rest of string after @ must be a number
                        end if
                    end repeat
                on error
                    set validEntry to false
                end try
            end if

            -- Alert and go back if invalid

            if validEntry is false then

                display dialog "\"" & levelsString & "\" is not a valid
entry! Try again." with title dialogTitle with icon 0 ¬
                    buttons {"OK"} default button "OK" giving up after 5

            else

                -- Final check that the levels have been interpreted
correctly

                set pleaseConfirm to ""
                repeat with i from 1 to countLevelsArray by 3
                    set eachRow to (item i of levelsArray) as number
                    set eachColumn to (item (i + 1) of levelsArray) as
```

```
number
                    set eachLevel to item (i + 2) of levelsArray
                    if eachRow is 0 then
                        if eachColumn is 0 then
                            set eachLine to "Master Level"
                        else
                            set eachLine to "Output " & eachColumn & "
Level"
                        end if
                    else
                        if eachColumn is 0 then
                            set eachLine to "Input " & eachRow & " Level"
                        else
                            set eachLine to "Crosspoint Level: Input " &
eachRow & " to Output " & eachColumn
                        end if
                    end if
                    if eachLevel does not start with "@" then
                        set eachLine to "> Add " & eachLevel & "dB to " &
eachLine
                    else
                        set eachLine to "> Set " & eachLine & " @ " & (rest
of characters of eachLevel as text) & "dB"
                    end if
                    set pleaseConfirm to pleaseConfirm & eachLine
                    if (i + 2) is not countLevelsArray then
                        set pleaseConfirm to pleaseConfirm & return
                    end if
                end repeat

                set goBack to button returned of (display dialog "Please
confirm you wish to adjust levels thus for the " & whichCuesString & ":" & ¬
                    return & return & pleaseConfirm with title dialogTitle
with icon 0 buttons {"Cancel", "No, no! Stop! That's not right! Go back!",
"OK"} ¬
                    default button "OK" cancel button "Cancel")
                if goBack is "No, no! Stop! That's not right! Go back!" then
                    set validEntry to false
                end if

            end if

        end repeat

        -- Now, to business

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
```

```
            set eachCue to item i of cuesToProcess
            try -- Skip over cues that don't take levels!
                repeat with j from 1 to countLevelsArray by 3
                    set eachRow to (item j of levelsArray) as number
                    set eachColumn to (item (j + 1) of levelsArray) as
number
                    set eachLevel to item (j + 2) of levelsArray
                    set currentLevel to getLevel eachCue row eachRow column
eachColumn ¬
                        -- This check will throw an error and exit the j
repeat if the cue doesn't take levels
                    if eachLevel does not start with "@" then
                        set eachLevel to currentLevel + eachLevel
                    else
                        set eachLevel to (rest of characters of eachLevel as
text) as number
                    end if
                    try -- This try will throw a detectable error if the
next line doesn't work
                        setLevel eachCue row eachRow column eachColumn db
eachLevel ¬
                            -- We're relying on QLab's ability to ignore
spurious levels like "-200" or "+50"
                        -- ###FIXME### As of 4.4.1, QLab appears to accept
any negative value for Min Volume Limit and hence here too
                    on error
                        set ohDear to true
                    end try
                end repeat
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustLevels

on adjustFileTarget(cuesToProcess, changeType, whichCuesString)

    tell application id "com.figure53.QLab.4"

        if changeType is "Change" then
            set promptHeader to "CHANGE File Target: start/end times
maintained…"
        else
```

```
            set promptHeader to "SET File Target: this process will not
maintain the start/end times!"
        end if

        set theTarget to choose file of type "public.audio" with prompt
promptHeader & return & return & ¬
            "Please select the File Target to set for the " &
whichCuesString & ":" without invisibles

        my startTheClock()

        set countCues to count cuesToProcess
        set checkTheFirst to true

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            if q type of eachCue is not "Group" then -- Setting File Target
on a Group Cue (or cue list) affects all Audio Cues in it!
                try -- Skip over cues that don't take File Targets!

                    if changeType is "Change" then
                        set currentStart to start time of eachCue
                        set currentEnd to end time of eachCue
                    end if

                    set file target of eachCue to theTarget ¬
                        -- QLab doesn't appear to throw any errors even if
the cue doesn't take a File Target, so no detection is possible

                    if checkTheFirst is true then -- Check the first one
worked (ie: didn't break the cue!) before going any further
                        if broken of eachCue is true then -- This protects
against (some) inappropriate files
                            set abortAbort to true
                            exit repeat
                        end if
                    end if

                    if changeType is "Change" then
                        set start time of eachCue to currentStart
                        set end time of eachCue to currentEnd
                    end if

                end try
                set checkTheFirst to false -- We need to check the first cue
that took a target, not the first cue we tried
            end if
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
```

```
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustFileTarget

on adjustSetName(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theName to my enterSomeText("Enter the name you wish to set for
the " & whichCuesString & ¬
            " (return an empty string to reset to default names):", "",
true)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q name of eachCue to theName
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSetName

on adjustSearchReplaceName(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set searchFor to my enterSomeText("Enter the search string you wish
to replace in the names of the " & whichCuesString & " (not case-
sensitive):", ¬
            "", false)
```

```
        set replaceWith to my enterSomeText("Enter the string with which you
wish to replace all occurrences of \"" & ¬
            searchFor & "\" in the names of the " & whichCuesString & ":",
"", true)

        my startTheClock()

        set countCues to count cuesToProcess
        set currentTIDs to AppleScript's text item delimiters

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentName to q list name of eachCue
            set AppleScript's text item delimiters to searchFor
            set searchedName to text items of currentName
            set AppleScript's text item delimiters to replaceWith
            set theName to searchedName as text
            set AppleScript's text item delimiters to currentTIDs
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q name of eachCue to theName
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSearchReplaceName

on adjustPrefixName(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set thePrefix to my enterSomeText("Enter the string you wish to add
to the beginning of the names of the " & whichCuesString & ¬
            " (include a space at the end if you expect one):", "", false)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
```

```
            set currentName to q list name of eachCue
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q name of eachCue to thePrefix & currentName
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustPrefixName

on adjustSuffixName(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theSuffix to my enterSomeText("Enter the string you wish to add
to the end of the names of the " & whichCuesString & ¬
            " (include a space at the beginning if you expect one):", "",
false)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentName to q list name of eachCue
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q name of eachCue to currentName & theSuffix
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()
```

```
    end tell

end adjustSuffixName

on adjustSeriesName(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set validEntry to false
        set previousBaseName to ""
        set previousStartNumber to ""
        set previousIncrement to ""
        set previousPadding to ""
        repeat until validEntry is true
            set baseName to my enterSomeText("Enter the base of the name
series you wish to set for the " & whichCuesString & ¬
                " (include a space at the end if you expect one):",
previousBaseName, false)
            set startNumber to my enterAnInteger("Enter an integer with
which to start the series:", previousStartNumber)
            set theIncrement to my enterAnInteger("Enter the increment for
each step (an integer):", previousIncrement)
            set thePadding to my
enterANumberWithRangeWithCustomButton("Enter the minimum number of digits –
an integer between 1 & 10 " & ¬
                "(eg: entering 2 will result in the series 01, 02, etc):",
previousPadding, 1, true, 10, true, true, {}, "OK")
            set previousBaseName to baseName
            set previousStartNumber to startNumber
            set previousIncrement to theIncrement
            set previousPadding to thePadding
            set counterConfirm1 to my padNumber(startNumber, thePadding)
            set pleaseConfirm1 to "> " & baseName & counterConfirm1
            set counterConfirm2 to my padNumber(startNumber + theIncrement,
thePadding)
            set pleaseConfirm2 to "> " & baseName & counterConfirm2
            set counterConfirm3 to my padNumber(startNumber + 2 *
theIncrement, thePadding)
            set pleaseConfirm3 to "> " & baseName & counterConfirm3
            set pleaseConfirm to pleaseConfirm1 & return & pleaseConfirm2 &
return & pleaseConfirm3 & return & "> …"
            set goBack to button returned of (display dialog "Please confirm
you wish to set the names as a series of this ilk for the " &
whichCuesString & ¬
                ":" & return & return & pleaseConfirm with title dialogTitle
with icon 0 buttons {"Cancel", "No, no! Stop! That's not right! Go back!",
"OK"} ¬
                default button "OK" cancel button "Cancel")
            if goBack is "OK" then
                set validEntry to true
            end if
```

```
        end repeat

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set theCounter to startNumber + (i - 1) * theIncrement
            set theCounter to my padNumber(theCounter, thePadding)
            set theName to baseName & theCounter
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q name of eachCue to theName
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSeriesName

on adjustPrefixNumber(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set thePrefix to my enterSomeText("Enter the string you wish to add
to the beginning of the Cue Numbers of the " & whichCuesString & ¬
            " (include a space at the end if you expect one):" & return &
return & ¬
            "(NB: if the Cue Number proposed already exists it won't be
changed.)", "", false)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentNumber to q number of eachCue
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q number of eachCue to thePrefix & currentNumber
```

```
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustPrefixNumber

on adjustSuffixNumber(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theSuffix to my enterSomeText("Enter the string you wish to add
to the end of the Cue Numbers of the " & whichCuesString & ¬
            " (include a space at the beginning if you expect one):" &
return & return & ¬
            "(NB: if the Cue Number proposed already exists it won't be
changed.)", "", false)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentNumber to q number of eachCue
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set q number of eachCue to currentNumber & theSuffix
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell
```

```
end adjustSuffixNumber

on adjustSeriesNumber(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set validEntry to false
        set previousBaseName to ""
        set previousStartNumber to ""
        set previousIncrement to ""
        set previousPadding to ""
        repeat until validEntry is true
            set baseName to my enterSomeText("Enter the base of the Cue
Number series you wish to set for the " & whichCuesString & ¬
                " (include a space at the end if you expect one):",
previousBaseName, false)
            set startNumber to my enterAnInteger("Enter an integer with
which to start the series:", previousStartNumber)
            set theIncrement to my enterAnInteger("Enter the increment for
each step (an integer):", previousIncrement)
            set thePadding to my
enterANumberWithRangeWithCustomButton("Enter the minimum number of digits –
an integer between 1 & 10 " & ¬
                "(eg: entering 2 will result in the series 01, 02, etc):",
previousPadding, 1, true, 10, true, true, {}, "OK")
            set previousBaseName to baseName
            set previousStartNumber to startNumber
            set previousIncrement to theIncrement
            set previousPadding to thePadding
            set counterConfirm1 to my padNumber(startNumber, thePadding)
            set pleaseConfirm1 to "> " & baseName & counterConfirm1
            set counterConfirm2 to my padNumber(startNumber + theIncrement,
thePadding)
            set pleaseConfirm2 to "> " & baseName & counterConfirm2
            set counterConfirm3 to my padNumber(startNumber + 2 *
theIncrement, thePadding)
            set pleaseConfirm3 to "> " & baseName & counterConfirm3
            set pleaseConfirm to pleaseConfirm1 & return & pleaseConfirm2 &
return & pleaseConfirm3 & return & "> …"
            set goBack to button returned of (display dialog ¬
                "Please confirm you wish to set the Cue Numbers as a series
of this ilk for the " & whichCuesString & ":" & return & return & ¬
                pleaseConfirm with title dialogTitle with icon 0 buttons
{"Cancel", "No, no! Stop! That's not right! Go back!", "OK"} ¬
                default button "OK" cancel button "Cancel")
            if goBack is "OK" then
                set validEntry to true
            end if
        end repeat

        my startTheClock()
```

```
        set countCues to count cuesToProcess
        set j to 0

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set existingCueNumber to true
            repeat while existingCueNumber is true
                set theCounter to startNumber + j * theIncrement
                set theCounter to my padNumber(theCounter, thePadding)
                set theName to baseName & theCounter
                try -- This try will throw a detectable error if the next
line doesn't work (hard to think of a reason why it wouldn't though!)
                    set q number of eachCue to theName
                    if q number of eachCue is theName then -- Check the
number has stuck; if it's already in use go round again
                        set existingCueNumber to false
                    else
                        set j to j + 1
                        set existingCueNumber to true
                    end if
                on error
                    set ohDear to true
                end try
            end repeat
            set j to j + 1
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSeriesNumber

on adjustClearNotes(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
```

```
                set notes of eachCue to ""
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustClearNotes

on adjustSetNotes(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theNotes to my enterSomeText("Enter the Notes you wish to set
for the " & whichCuesString & ":", "", false)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set notes of eachCue to theNotes
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSetNotes

on adjustSearchReplaceNotes(cuesToProcess, whichCuesString)
```

```
    tell application id "com.figure53.QLab.4"

        set searchFor to my enterSomeText("Enter the search string you wish
to replace in the Notes of the " & whichCuesString & " (not case-
sensitive):", "", false)
        set replaceWith to my enterSomeText("Enter the string with which
wish to replace all occurrences of \"" & ¬
            searchFor & "\" in the Notes of the " & whichCuesString & ":",
"", true)

        my startTheClock()

        set countCues to count cuesToProcess
        set currentTIDs to AppleScript's text item delimiters

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentNotes to notes of eachCue
            set AppleScript's text item delimiters to searchFor
            set searchedNotes to text items of currentNotes
            set AppleScript's text item delimiters to replaceWith
            set theNotes to searchedNotes as text
            set AppleScript's text item delimiters to currentTIDs
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set notes of eachCue to theNotes
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSearchReplaceNotes

on adjustPrefixNotes(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set thePrefix to my enterSomeText("Enter the string you wish to add
to the beginning of the Notes of the " & whichCuesString & ¬
            " (include a space at the end if you expect one):", "", false)

        my startTheClock()
```

```
        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentNotes to notes of eachCue
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set notes of eachCue to thePrefix & currentNotes
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustPrefixNotes

on adjustSuffixNotes(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theSuffix to my enterSomeText("Enter the string you wish to add
to the end of the Notes of the " & whichCuesString & ¬
            " (include a space at the beginning if you expect one):", "",
false)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set currentNotes to notes of eachCue
            try -- This try will throw a detectable error if the next line
doesn't work (hard to think of a reason why it wouldn't though!)
                set notes of eachCue to currentNotes & theSuffix
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
```

```
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSuffixNotes

on adjustSetTime(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceTimesParameter = {"Pre Wait", "Duration", "Post Wait"}

    tell application id "com.figure53.QLab.4"

        if theParameter is item 3 of subChoiceTimesParameter then -- Special
option to set Post Waits to the same time as the Duration
            set specialCase to "Set to Duration"
        else
            set specialCase to {}
        end if

        set theTime to my enterATimeWithCustomButton("Enter the time you
wish to set as the " & theParameter & ¬
            " for the " & whichCuesString & " (seconds or
minutes:seconds):", "", specialCase)

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the appropriate
line doesn't work
                if theParameter is item 1 of subChoiceTimesParameter then
                    set pre wait of eachCue to theTime
                else if theParameter is item 2 of subChoiceTimesParameter
then
                    set duration of eachCue to theTime
                else if theParameter is item 3 of subChoiceTimesParameter
then
                    if theTime is specialCase then
                        set theDuration to duration of eachCue
                        set post wait of eachCue to theDuration
                    else
                        set post wait of eachCue to theTime
                    end if
                end if
            on error
                set ohDear to true
```

```
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSetTime

on adjustScaleTime(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceTimesParameter = {"Pre Wait", "Duration", "Post Wait"}

    tell application id "com.figure53.QLab.4"

        set theMultiplicand to my enterARatio("Enter the ratio with which
you wish to scale the " & theParameter & ¬
            " for the " & whichCuesString & " (eg: 1.1 will make them 10%
longer):", "")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the appropriate
line doesn't work
                if theParameter is item 1 of subChoiceTimesParameter then
                    set currentTime to pre wait of eachCue
                    set pre wait of eachCue to currentTime * theMultiplicand
                else if theParameter is item 2 of subChoiceTimesParameter
then
                    set currentTime to duration of eachCue
                    set duration of eachCue to currentTime * theMultiplicand
                else if theParameter is item 3 of subChoiceTimesParameter
then
                    set currentTime to post wait of eachCue
                    set post wait of eachCue to currentTime *
theMultiplicand
                end if
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
```

```
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustScaleTime

on adjustAddSubractTime(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceTimesParameter = {"Pre Wait", "Duration", "Post Wait"}

    tell application id "com.figure53.QLab.4"

        set theAddend to my enterANumber("Enter the number of seconds you
wish to add to " & theParameter & " for the " & whichCuesString & ":", "")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the appropriate
line doesn't work
                if theParameter is item 1 of subChoiceTimesParameter then
                    set currentTime to pre wait of eachCue
                    set pre wait of eachCue to currentTime + theAddend
                else if theParameter is item 2 of subChoiceTimesParameter
then
                    set currentTime to duration of eachCue
                    set duration of eachCue to currentTime + theAddend
                else if theParameter is item 3 of subChoiceTimesParameter
then
                    set currentTime to post wait of eachCue
                    set post wait of eachCue to currentTime + theAddend
                end if
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()
```

```
    end tell

end adjustAddSubractTime

on adjustSetMIDICommand(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theParameter to my pickFromList(subChoiceMIDICommand, "Set the
MIDI Command of the selected cues to:")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- Skip over cues that don't take MIDI!
                if message type of eachCue is not voice then -- This will
throw an error if eachCue isn't a MIDI Cue
                    error -- This rejects MSC & SysEx cues
                end if
                try -- This try will throw a detectable error if the
appropriate line doesn't work
                    if theParameter is item 1 of subChoiceMIDICommand then
                        set command of eachCue to note_on
                    else if theParameter is item 2 of subChoiceMIDICommand
then
                        set command of eachCue to note_off
                    else if theParameter is item 3 of subChoiceMIDICommand
then
                        set command of eachCue to program_change
                    else if theParameter is item 4 of subChoiceMIDICommand
then
                        set command of eachCue to control_change
                    else if theParameter is item 5 of subChoiceMIDICommand
then
                        set command of eachCue to key_pressure
                    else if theParameter is item 6 of subChoiceMIDICommand
then
                        set command of eachCue to channel_pressure
                    else if theParameter is item 7 of subChoiceMIDICommand
then
                        set command of eachCue to pitch_bend
                    end if
                on error
                    set ohDear to true
                end try
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
```

```
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSetMIDICommand

on adjustSetMIDI(cuesToProcess, theParameter, whichCuesString) -- This is
the only one that handles "Channel"

    -- subChoiceMIDIParameter = {"Command", "Channel", "Byte One", "Byte
Two", "Byte Combo", "End Value"}

    tell application id "com.figure53.QLab.4"

        if theParameter is item 2 of subChoiceMIDIParameter then
            set theMin to 1
            set theMax to 16
        else if theParameter is item 5 of subChoiceMIDIParameter then
            set theMin to -8192
            set theMax to 8191
        else
            set theMin to 0
            set theMax to 127
        end if

        set theInteger to my enterANumberWithRangeWithCustomButton("Enter
the value to which you wish to set the " & theParameter & ¬
            " for the " & whichCuesString & ":", "", theMin, true, theMax,
true, true, {}, "OK")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- Skip over cues that don't take MIDI!
                if message type of eachCue is not voice then -- This will
throw an error if eachCue isn't a MIDI Cue
                    error -- This rejects MSC & SysEx cues
                end if
                try -- This try will throw a detectable error if the
appropriate line doesn't work
                    if theParameter is item 2 of subChoiceMIDIParameter then
                        set channel of eachCue to theInteger
```

```
                    else if theParameter is item 3 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set byte one of eachCue to theInteger
                        end if
                    else if theParameter is item 4 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set byte two of eachCue to theInteger
                        end if
                    else if theParameter is item 5 of subChoiceMIDIParameter
then
                        if command of eachCue is pitch_bend then
                            set byte combo of eachCue to theInteger + 8192 -
- Pitch bend of 0 in the Inspector is reported to AppleScript as 8192
                        end if
                    else if theParameter is item 6 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set end value of eachCue to theInteger
                        else
                            set end value of eachCue to theInteger + 8192
                        end if
                    end if
                on error
                    set ohDear to true
                end try
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSetMIDI

on adjustScaleMIDI(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceMIDIParameter = {"Command", "Channel", "Byte One", "Byte
Two", "Byte Combo", "End Value"}

    tell application id "com.figure53.QLab.4"

        set theMultiplicand to my enterARatio("Enter the ratio with which
you wish to scale the " & theParameter & ¬
            " for the " & whichCuesString & " (eg: 1.1 will make them 10%
```

```
larger):", "")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- Skip over cues that don't take MIDI!
                if message type of eachCue is not voice then -- This will
throw an error if eachCue isn't a MIDI Cue
                    error -- This rejects MSC & SysEx cues
                end if
                try -- This try will throw a detectable error if the
appropriate line doesn't work
                    if theParameter is item 3 of subChoiceMIDIParameter then
                        if command of eachCue is not pitch_bend then
                            set currentValue to byte one of eachCue
                            set byte one of eachCue to currentValue *
theMultiplicand
                        end if
                    else if theParameter is item 4 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set currentValue to byte two of eachCue
                            set byte two of eachCue to currentValue *
theMultiplicand
                        end if
                    else if theParameter is item 5 of subChoiceMIDIParameter
then
                        if command of eachCue is pitch_bend then
                            set currentValue to (byte combo of eachCue) -
8192
                            set byte combo of eachCue to currentValue *
theMultiplicand + 8192
                        end if
                    else if theParameter is item 6 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set currentValue to end value of eachCue
                            set end value of eachCue to currentValue *
theMultiplicand
                        else
                            set currentValue to ((end value of eachCue) -
8192)
                            set end value of eachCue to (currentValue *
theMultiplicand) + 8192
                        end if
                    end if
                on error
                    set ohDear to true
```

```
                end try
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustScaleMIDI

on adjustAddSubractMIDI(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceMIDIParameter = {"Command", "Channel", "Byte One", "Byte
Two", "Byte Combo", "End Value"}

    tell application id "com.figure53.QLab.4"

        set theAddend to my enterAnInteger("Enter the integer you wish to
add to " & theParameter & " for the " & whichCuesString & ":", "")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- Skip over cues that don't take MIDI!
                if message type of eachCue is not voice then -- This will
throw an error if eachCue isn't a MIDI Cue
                    error -- This rejects MSC & SysEx cues
                end if
                try -- This try will throw a detectable error if the
appropriate line doesn't work
                    if theParameter is item 3 of subChoiceMIDIParameter then
                        set currentValue to byte one of eachCue
                        set byte one of eachCue to currentValue + theAddend
                    else if theParameter is item 4 of subChoiceMIDIParameter
then
                        set currentValue to byte two of eachCue
                        set byte two of eachCue to currentValue + theAddend
                    else if theParameter is item 5 of subChoiceMIDIParameter
then
                        set currentValue to byte combo of eachCue
                        set byte combo of eachCue to currentValue +
theAddend
                    else if theParameter is item 6 of subChoiceMIDIParameter
```

```
then
                        set currentValue to end value of eachCue
                        set end value of eachCue to currentValue + theAddend
                    end if
                on error
                    set ohDear to true
                end try
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustAddSubractMIDI

on adjustSeriesMIDI(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceMIDIParameter = {"Command", "Channel", "Byte One", "Byte
Two", "Byte Combo", "End Value"}

    tell application id "com.figure53.QLab.4"

        if theParameter is item 5 of subChoiceMIDIParameter then
            set theMin to -8192
            set theMax to 8191
        else
            set theMin to 0
            set theMax to 127
        end if

        set startNumber to my enterANumberWithRangeWithCustomButton("Enter
the value to which you wish to set the " & theParameter & ¬
            " of the first of the " & whichCuesString & ":", "", theMin,
true, theMax, true, true, {}, "OK")
        set theIncrement to my enterAnInteger("Enter the increment for each
step:", "")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            set theCounter to startNumber + (i - 1) * theIncrement -- We're
relying on QLab to hit a ceiling/floor for this
```

```
            try -- Skip over cues that don't take MIDI!
                if message type of eachCue is not voice then -- This will
throw an error if eachCue isn't a MIDI Cue
                    error -- This rejects MSC & SysEx cues
                end if
                try -- This try will throw a detectable error if the
appropriate line doesn't work
                    if theParameter is item 2 of subChoiceMIDIParameter then
                        set channel of eachCue to theCounter
                    else if theParameter is item 3 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set byte one of eachCue to theCounter
                        end if
                    else if theParameter is item 4 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set byte two of eachCue to theCounter
                        end if
                    else if theParameter is item 5 of subChoiceMIDIParameter
then
                        if command of eachCue is pitch_bend then
                            set byte combo of eachCue to theCounter + 8192 -
- Pitch bend of 0 in the Inspector is reported to AppleScript as 8192
                        end if
                    else if theParameter is item 6 of subChoiceMIDIParameter
then
                        if command of eachCue is not pitch_bend then
                            set end value of eachCue to theCounter
                        else
                            set end value of eachCue to theCounter + 8192
                        end if
                    end if
                on error
                    set ohDear to true
                end try
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSeriesMIDI

on adjustDeviceID(cuesToProcess, whichCuesString)
```

```
    tell application id "com.figure53.QLab.4"

        set theParameter to my enterANumberWithRangeWithCustomButton("Enter
the MSC Device ID you wish to set for the " & whichCuesString & ¬
            " (an integer from 0 to 127):", "", 0, true, 127, true, true,
{}, "OK")

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            if message type of eachCue is msc then
                try -- This try will throw a detectable error if the
appropriate line doesn't work (hard to think of a reason why it wouldn't
though!)
                    set deviceID of eachCue to theParameter
                on error
                    set ohDear to true
                end try
            end if
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustDeviceID

on adjustAutoload(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceAutoload = {"On", "Off"}

    tell application id "com.figure53.QLab.4"

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the appropriate
line doesn't work (hard to think of a reason why it wouldn't though!)
                if theParameter is item 1 of subChoiceAutoload then
```

```
                    set autoload of eachCue to true
                else if theParameter is item 2 of subChoiceAutoload then
                    set autoload of eachCue to false
                end if
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustAutoload

on adjustArmed(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceArmed = {"Armed", "Disarmed"}

    tell application id "com.figure53.QLab.4"

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the appropriate
line doesn't work (hard to think of a reason why it wouldn't though!)
                if theParameter is item 1 of subChoiceArmed then
                    set armed of eachCue to true
                else if theParameter is item 2 of subChoiceArmed then
                    set armed of eachCue to false
                end if
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()
```

```
    end tell

end adjustArmed

on adjustContinueMode(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceContinueMode = {"Do not continue", "Auto-continue", "Auto-
follow"}

    tell application id "com.figure53.QLab.4"

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- This try will throw a detectable error if the appropriate
line doesn't work (hard to think of a reason why it wouldn't though!)
                if theParameter is item 1 of subChoiceContinueMode then
                    set continue mode of eachCue to do_not_continue
                else if theParameter is item 2 of subChoiceContinueMode then
                    set continue mode of eachCue to auto_continue
                else if theParameter is item 3 of subChoiceContinueMode then
                    set continue mode of eachCue to auto_follow
                end if
            on error
                set ohDear to true
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustContinueMode

on adjustMode(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceMode = {"Timeline - Start all children
simultaneously","Start first child and enter into group", "Start first child
and go to next cue" ,
    (* "Start random child and go to next cue"} *)

    tell application id "com.figure53.QLab.4"
```

```
        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            try -- Skip over cues that aren't Groups
                if mode of eachCue is not cue_list then -- Don't adjust cue
lists!
                    try -- This try will throw a detectable error if the
appropriate line doesn't work
                        if theParameter is item 1 of subChoiceMode then
                            set mode of eachCue to timeline
                        else if theParameter is item 2 of subChoiceMode then
                            set mode of eachCue to fire_first_enter_group
                        else if theParameter is item 3 of subChoiceMode then
                            set mode of eachCue to fire_first_go_to_next_cue
                        else if theParameter is item 4 of subChoiceMode then
                            set mode of eachCue to fire_random
                        end if
                    on error
                        set ohDear to true
                    end try
                end if
            end try
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustMode

on adjustPatch(cuesToProcess, whichCuesString)

    tell application id "com.figure53.QLab.4"

        set theParameter to my enterANumberWithRangeWithCustomButton("Enter
the patch you wish to set for the " & whichCuesString & ¬
            " (1-8, 1-16 for Network Cues):", "", 1, true, false, false,
true, {}, "OK")

        my startTheClock()

        set countCues to count cuesToProcess
```

```
        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            if q type of eachCue is in {"Audio", "Mic", "Video", "Network",
"MIDI", "MIDI File", "Timecode"} then
                try -- This try will throw a detectable error if the
appropriate line doesn't work (hard to think of a reason why it wouldn't
though!)
                    set patch of eachCue to theParameter
                on error
                    set ohDear to true
                end try
            end if
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustPatch

on adjustCameraPatch(cuesToProcess, theParameter, whichCuesString)

    -- subChoiceCameraPatch = {1, 2, 3, 4, 5, 6, 7, 8}

    tell application id "com.figure53.QLab.4"

        my startTheClock()

        set countCues to count cuesToProcess

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            if q type of eachCue is "Camera" then
                try -- This try will throw a detectable error if the
appropriate line doesn't work (hard to think of a reason why it wouldn't
though!)
                    set camera patch of eachCue to theParameter
                on error
                    set ohDear to true
                end try
            end if
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
```

```
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustCameraPatch

on adjustSearchReplaceCommandText(cuesToProcess, whichCuesString,
instrumentNamesONLY)

    tell application id "com.figure53.QLab.4"

        if instrumentNamesONLY is true then
            set searchFor to my enterSomeText("Enter the search string you
wish to replace in the Instrument Names in the Command Text of the " & ¬
                whichCuesString & " (not case-sensitive):", "", false)
            set replaceWith to my enterSomeText("Enter the string with which
wish to replace all occurrences of \"" & ¬
                searchFor & "\" in Instrument Names in the the Command Text
of the " & whichCuesString & ":", "", true)
            set searchFor to searchFor & " = "
            set replaceWith to replaceWith & " = "
        else
            set searchFor to my enterSomeText("Enter the search string you
wish to replace in the Command Text of the " & whichCuesString & ¬
                " (not case-sensitive):", "", false)
            set replaceWith to my enterSomeText("Enter the string with which
wish to replace all occurrences of \"" & ¬
                searchFor & "\" in the Command Text of the " &
whichCuesString & ":", "", true)
        end if

        my startTheClock()

        set countCues to count cuesToProcess
        set currentTIDs to AppleScript's text item delimiters

        repeat with i from 1 to countCues
            set eachCue to item i of cuesToProcess
            if q type of eachCue is "Light" then
                try -- This try will throw a detectable error if something
doesn't work
                    set currentCommandText to command text of eachCue
                    if currentCommandText is not missing value then -- Skip
cues that are empty
                        set AppleScript's text item delimiters to searchFor
                        set searchedCommandText to text items of
currentCommandText
                        set AppleScript's text item delimiters to
replaceWith
```

```
                        set theCommandText to searchedCommandText as text
                        set AppleScript's text item delimiters to
currentTIDs
                        set command text of eachCue to theCommandText
                    end if
                on error
                    set ohDear to true
                end try
            end if
            if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                my countdownTimer(i, countCues, whichCuesString)
            end if
        end repeat

        my finishedDialogBespoke()

    end tell

end adjustSearchReplaceCommandText

(* === TIME === *)

on makeMSS(howLong) -- [Shared subroutine]
    set howManyMinutes to howLong div 60
    set howManySeconds to howLong mod 60 div 1
    return (howManyMinutes as text) & ":" & my padNumber(howManySeconds, 2)
end makeMSS

on makeNiceT(howLong) -- [Shared subroutine]
    if howLong < 1 then
        return "less than a second"
    end if
    set howManyHours to howLong div 3600
    if howManyHours is 0 then
        set hourString to ""
    else if howManyHours is 1 then
        set hourString to "1 hour"
    else
        set hourString to (howManyHours as text) & " hours"
    end if
    set howManyMinutes to howLong mod 3600 div 60
    if howManyMinutes is 0 then
        set minuteString to ""
    else if howManyMinutes is 1 then
        set minuteString to "1 minute"
    else
        set minuteString to (howManyMinutes as text) & " minutes"
    end if
    set howManySeconds to howLong mod 60 div 1
```

```
    if howManySeconds is 0 then
        set secondString to ""
    else if howManySeconds is 1 then
        set secondString to "1 second"
    else
        set secondString to (howManySeconds as text) & " seconds"
    end if
    set theAmpersand to ""
    if hourString is not "" then
        if minuteString is not "" and secondString is not "" then
            set theAmpersand to ", "
        else if minuteString is not "" or secondString is not "" then
            set theAmpersand to " and "
        end if
    end if
    set theOtherAmpersand to ""
    if minuteString is not "" and secondString is not "" then
        set theOtherAmpersand to " and "
    end if
    return hourString & theAmpersand & minuteString & theOtherAmpersand &
secondString
end makeNiceT

on makeSecondsFromM_S(howLong) -- [Shared subroutine]
    try
        set currentTIDs to AppleScript's text item delimiters
        set AppleScript's text item delimiters to ":"
        set theMinutes to first text item of howLong
        set theSeconds to rest of text items of howLong as text
        set AppleScript's text item delimiters to currentTIDs
        return theMinutes * 60 + theSeconds
    on error
        return false
    end try
end makeSecondsFromM_S

(* === TEXT WRANGLING === *)

on padNumber(theNumber, minimumDigits) -- [Shared subroutine]
    set paddedNumber to theNumber as text
    repeat while (count paddedNumber) < minimumDigits
        set paddedNumber to "0" & paddedNumber
    end repeat
    return paddedNumber
end padNumber
```

General

Group selected cue(s)

Put selected cue(s) in a new Group Cue:

WARNING!!!! This script will move selected cues out of any Group Cues they are in and into the new Group Cue. If you want a Group Cue to remain intact when it is moved, DO NOT include its children in the selection! Also, remember that new cues are inserted into the cue list after the last cue that was selected (which is not always the same as the cue at the bottom of the selection); this will dictate where the new Group Cue is made, and hence what is moved into it – as the script can't move an existing Group Cue inside the new Group Cue if the new Group Cue is made within the existing Group Cue... It is subtly different to the built-in behaviour when making a new Group Cue with a selection, not least because it acts on single selected cues! The new Group Cue's settings will follow the Workspace Preferences, by the way. (Doesn't fire in a cart!)

IT IS VERY EASY TO CONSTRUCT ELABORATE NESTS OF GROUPS THAT WILL CAUSE THIS SCRIPT TO HANG QLAB!

```
tell front workspace

    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!

    set selectedCues to (selected as list)

    if (count selectedCues) is not 0 then

        set selected to last item of selectedCues -- Protect against default
behaviour
        make type "Group"
        set groupCue to last item of (selected as list)
        set groupCueIsIn to parent of groupCue

        repeat with eachCue in selectedCues
            if contents of eachCue is not groupCueIsIn then -- Skip a Group
Cue that contains the new Group Cue
                set eachCueID to uniqueID of eachCue
                try
                    move cue id eachCueID of parent of eachCue to end of
groupCue
                end try
            end if
        end repeat

    end if

end tell
```

Group selected cue(s) with number & notes

Put selected cue(s) in a new Group Cue, retaining Cue Number & Notes from first cue:

WARNING!!!! This script will move selected cues out of any Group Cues they are in and into the new Group Cue. If you want a Group Cue to remain intact when it is moved, DO NOT include its children in the selection! Also, remember that new cues are inserted into the cue list after the last cue that was selected (which is not always the same as the cue at the bottom of the selection); this will dictate where the new Group Cue is made, and hence what is moved into it – as the script can't move an existing Group Cue inside the new Group Cue if the new Group Cue is made within the existing Group Cue... It is subtly different to the built-in behaviour when making a new Group Cue with a selection, not least because it acts on single selected cues! The new Group Cue's settings will follow the Workspace Preferences, by the way. (Doesn't fire in a cart!)

IT IS VERY EASY TO CONSTRUCT ELABORATE NESTS OF GROUPS THAT WILL CAUSE THIS SCRIPT TO HANG QLAB!

```
tell front workspace

    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!

    set selectedCues to (selected as list)

    if (count selectedCues) is not 0 then

        set selected to last item of selectedCues -- Protect against default
behaviour
        make type "Group"
        set groupCue to last item of (selected as list)
        set groupCueIsIn to parent of groupCue

        set cueNumber to q number of first item of selectedCues
        set q number of first item of selectedCues to ""
        set q number of groupCue to cueNumber

        set cueNames to {}
        repeat with i from 1 to count selectedCues
            set eachName to q list name of item i of selectedCues
            if eachName is not "" then
                set end of cueNames to eachName
            end if
        end repeat
        set currentTIDs to AppleScript's text item delimiters
        set AppleScript's text item delimiters to " & "
        set q name of groupCue to cueNames as text
        set AppleScript's text item delimiters to currentTIDs

        set cueNotes to notes of first item of selectedCues
        set notes of first item of selectedCues to ""
        set notes of groupCue to cueNotes

        repeat with eachCue in selectedCues
```

```
            if contents of eachCue is not groupCueIsIn then -- Skip a Group
Cue that contains the new Group Cue
                set eachCueID to uniqueID of eachCue
                try
                    move cue id eachCueID of parent of eachCue to end of
groupCue
                end try
            end if
        end repeat

    end if

end tell
```

Convert Group Cue

Convert Group Cues between “Start first child and go to next cue” sequences and “Timeline” with independent Pre Waits; IT IS NOT GUARANTEED TO COVER EVERY POSSIBLE SCENARIO SO USE IT **AT YOUR OWN RISK**! In particular, it will not work if the children of a “Timeline” Group Cue are not in the order in which their Pre Waits will cause them to fire (ie: with each child's Pre Wait greater than or equal to that of the previous child)…

```
tell front workspace
    try
        set selectedCue to last item of (selected as list)
        if q type of selectedCue is "Group" then
            if mode of selectedCue is fire_first_go_to_next_cue then
                set theCues to cues of selectedCue
                set deltaTime to 0
                repeat with eachCue in theCues
                    set eachPre to pre wait of eachCue
                    set deltaTime to deltaTime + eachPre
                    set pre wait of eachCue to deltaTime
                    if continue mode of eachCue is auto_follow then
                        set eachPost to duration of eachCue
                    else
                        set eachPost to post wait of eachCue
                    end if
                    set deltaTime to deltaTime + eachPost
                    set post wait of eachCue to 0
                    set continue mode of eachCue to do_not_continue
                end repeat
                set mode of selectedCue to timeline -- Can't set continue
modes while mode is timeline, so have to do this here…
                if post wait of selectedCue is duration of selectedCue then
-- Clear Post Wait if it looks like it's been set by "effective duration"
script
                    set post wait of selectedCue to 0
                end if
            else if mode of selectedCue is timeline then
```

```
                set theCues to cues of selectedCue
                set previousPre to 0
                set mode of selectedCue to fire_first_go_to_next_cue
                repeat with eachCue in theCues
                    set eachPre to pre wait of eachCue
                    set deltaTime to eachPre - previousPre
                    set previousPre to eachPre
                    set pre wait of eachCue to deltaTime
                    if contents of eachCue is not last item of theCues then
                        set continue mode of eachCue to auto_continue
                    end if
                end repeat
            end if
        end if
    end try
end tell
```

Set target

Set target of selected cue to cue above (for a selected Fade Cue, the script will find the closest preceding Group, Audio, Mic, Video, Camera or Text Cue (rather than just the cue above); likewise, Devamp Cues will look for the closest Audio or Video Cue):

```
-- Declarations

set simpleCases to {"Start", "Stop", "Pause", "Load", "Reset", "GoTo",
"Target", "Arm", "Disarm"}
set specialCases to {"Fade", "Devamp"}
set acceptableTargets to {{"Group", "Audio", "Mic", "Video", "Camera",
"Text"}, {"Audio", "Video"}}

-- Main routine

tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set originalCue to last item of (selected as list)
        set originalType to q type of originalCue
        if originalType is in simpleCases then
            set targetCue to cue before originalCue
            set cue target of originalCue to targetCue
            set targetName to q list name of targetCue
            if targetName is "" then
                set targetName to q display name of targetCue
            end if
            set q name of originalCue to originalType & ": " & targetName
        else if originalType is in specialCases then
            repeat with i from 1 to count specialCases
                if originalType is item i of specialCases then
                    set acceptableTypes to item i of acceptableTargets
```

```
                    exit repeat
                end if
            end repeat
            set foundType to ""
            set checkedCue to originalCue
            repeat while foundType is not in acceptableTypes
                set targetCue to cue before checkedCue
                set foundType to q type of targetCue
                if targetCue is first item of cues of current cue list then
-- Protect against infinite loop if no acceptable target found
                    exit repeat
                end if
                set checkedCue to targetCue
            end repeat
            if foundType is in acceptableTypes then -- Don't change the
target if we've gone all the way up the cue list without finding one
                set cue target of originalCue to targetCue
                set targetName to q list name of targetCue
                if targetName is "" then
                    set targetName to q display name of targetCue
                end if
                set q name of originalCue to originalType & ": " &
targetName
            end if
        end if
    end try
end tell
```

Reveal target file

Reveal selected cue's target in Finder (although this functionality is built-in, the button takes some locating...):

```
tell front workspace
    try -- This protects against no selection (can't get last item of
(selected as list))
        set selectedCue to last item of (selected as list)
        set fileTarget to file target of selectedCue
        tell application "Finder"
            reveal fileTarget
            activate
        end tell
    end try
end tell
```

Import files

Choose multiple audio, video and MIDI files and add them to the workspace as new cues; the original

files will be copied to the appropriate subfolder next to the workspace, unless they already exist there (in which case the existing files will be used); the script can be adjusted to enforce copying only for non-local files, eg: those on network servers:

```
set userCopyOnImport to true -- Set this to false if you do not want the
files to be copied to the workspace folder
set userForceCopyFromNetwork to true -- Set this to false if you do not want
files that aren't on local drives to be copied automatically
set userReassuranceThreshold to 5 -- When the number of files imported is
greater than this variable, a dialog will let you know that the process is
complete

-- ###FIXME### Should audioFileTypes include "public.ulaw-audio"?
-- ###FIXME### Is videoFileTypes a sufficiently exhaustive list? Are any of
the file types not supported by QLab?

-- Declarations

global dialogTitle, sharedPath
set dialogTitle to "Import files"

set audioFileTypes to {"com.apple.coreaudio-format", "com.apple.m4a-audio",
"com.microsoft.waveform-audio", "public.aifc-audio", "public.aiff-audio", ¬
    "public.audio", "public.mp3", "public.mpeg-4-audio"} -- This list
deliberately excludes "com.apple.protected-mpeg-4-audio" to protect against
(* old DRM-restricted iTunes files *)
set videoFileTypes to {"com.adobe.photoshop-image", "com.apple.icns",
"com.apple.macpaint-image", "com.apple.pict", "com.apple.quicktime-image", ¬
    "com.apple.quicktime-movie", "public.3gpp", "public.3gpp2",
"public.avi", "public.camera-raw-image", "public.image", "public.jpeg",
"public.jpeg-2000", ¬
    "public.movie", "public.mpeg", "public.mpeg-4", "public.png",
"public.tiff", "public.video", "public.xbitmap-image"}
set midiFileTypes to {"public.midi-audio"}

(* cf:
https://developer.apple.com/library/content/documentation/Miscellaneous/Refe
rence/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html *)

set theFileTypes to {audioFileTypes, videoFileTypes, midiFileTypes}
set foldersExist to {null, null, null}
set theSubfolders to {"audio", "video", "midi file"}
set theCueTypes to {"Audio", "Video", "MIDI File"}

-- Main routine

tell front workspace

    -- Establish the path to the current workspace

    set workspacePath to path
```

```
    if workspacePath is missing value then
        display dialog "The current workspace has not yet been saved
anywhere." with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK" giving up after 5
        return
    end if

    -- Get the path that should prefix all media file paths

    tell application "System Events"
        set sharedPath to path of container of file workspacePath
    end tell

    -- Choose the files to import

    set newTargets to choose file of type {"public.image",
"public.audiovisual-content"} ¬
        with prompt "Please select one or more audio, video or MIDI files:"
with multiple selections allowed

    -- Import them

    repeat with eachFile in newTargets

        tell application "System Events"
            set eachType to type identifier of eachFile
            set eachName to name of eachFile
            if userForceCopyFromNetwork is true then -- Only check file's
locality if it will be relevant
                set fileIsLocal to local volume of disk (volume of eachFile)
            else
                set fileIsLocal to true
            end if
        end tell

        set eachTarget to eachFile -- This variable will be updated if the
file is copied

        -- Work through the three types of cues that will be processed

        repeat with i from 1 to 3

            if eachType is in contents of item i of theFileTypes then

                if (userCopyOnImport is true) or (userForceCopyFromNetwork
is true and fileIsLocal is false) then

                    -- If copying is specified by the user definitions then…

                    -- Check for appropriate subfolder next to workspace and
make it if it doesn't exist
```

```
                    if item i of foldersExist is null then
                        set item i of foldersExist to my checkForFolder(item
i of theSubfolders)
                        if item i of foldersExist is false then
                            my makeFolder(item i of theSubfolders)
                        end if
                    end if

                    -- If the file is not already in place, copy it to the
appropriate subfolder

                    if my checkForFile(item i of theSubfolders, eachName) is
false then
                        my copyFileViaFinder(item i of theSubfolders,
eachFile)
                    end if

                    set eachTarget to sharedPath & item i of theSubfolders &
":" & eachName

                end if

                -- Make an appropriate cue

                make type item i of theCueTypes
                set newCue to last item of (selected as list)
                set file target of newCue to eachTarget

                exit repeat

            end if

        end repeat

    end repeat

end tell

if (count newTargets) > userReassuranceThreshold then
    display dialog "Done." with title dialogTitle with icon 1 buttons {"OK"}
default button "OK" giving up after 5
end if

-- Subroutines

(* === FILE WRANGLING === *)

on checkForFile(theSuffix, theName) -- [Shared subroutine]
    tell application "System Events"
        return exists file (sharedPath & theSuffix & ":" & theName)
```

```
    end tell
end checkForFile

on checkForFolder(theSuffix) -- [Shared subroutine]
    tell application "System Events"
        return exists folder (sharedPath & theSuffix)
    end tell
end checkForFolder

on makeFolder(theFolder) -- [Shared subroutine]
    tell application "Finder"
        make new folder at sharedPath with properties {name:theFolder}
    end tell
end makeFolder

on copyFileViaFinder(theSuffix, theFile)
    (* NB: by using the Finder the usual file-copy progress window is
invoked, which may be more reassuring than the faceless
    'do shell script "cp -p " & quoted form of POSIX path of theFile & " " &
quoted form of POSIX path of (sharedPath & theSuffix & ":" & theName)'
    - which may look like a freeze (the -p flag copies every property of a
file; "theName" would need to be passed to the subroutine to implement this)
*)
    tell application "Finder"
        duplicate theFile to folder (sharedPath & theSuffix)
    end tell
end copyFileViaFinder
```

Localise media

Copy the audio, video and MIDI files referenced by the selected cue(s) to the appropriate subfolder next to the workspace (unless they already exist there, in which case the existing files will be used), keeping start/end times:

```
-- QLab retains slice points within the duration of a new File Target but
resets the start & end times (this script maintains start & end times)

-- Declarations

global dialogTitle, sharedPath
set dialogTitle to "Localise media"

set foldersExist to {null, null, null}
set theSubfolders to {"audio", "video", "midi file"}
set theCueTypes to {"Audio", "Video", "MIDI File"}

-- Main routine

tell front workspace
```

```
    -- Establish the path to the current workspace

    set workspacePath to path
    if workspacePath is missing value then
        display dialog "The current workspace has not yet been saved
anywhere." with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK" giving up after 5
        return
    end if

    -- Get the path that should prefix all media file paths

    tell application "System Events"
        set sharedPath to path of container of file workspacePath
    end tell

    -- Work through the selected cues

    display dialog "One moment caller…" with title dialogTitle with icon 1
buttons {"OK"} default button "OK" giving up after 1

    repeat with eachCue in (selected as list)

        set eachType to q type of eachCue

        if eachType is in theCueTypes then

            -- Identify which item of the declared lists to use

            repeat with i from 1 to 3
                if eachType is item i of theCueTypes then
                    set eachIndice to i
                    set eachSubfolder to item eachIndice of theSubfolders
                    exit repeat
                end if
            end repeat

            -- Get the existing target (the try protects against missing
File Targets)

            try

                set eachFile to file target of eachCue as alias
                tell application "System Events"
                    set eachName to name of eachFile
                end tell

                -- Check for appropriate subfolder next to workspace and
make it if it doesn't exist

                if item eachIndice of foldersExist is null then
```

```
                    set item eachIndice of foldersExist to my
checkForFolder(eachSubfolder)
                    if item eachIndice of foldersExist is false then
                        my makeFolder(eachSubfolder)
                    end if
                end if

                -- If the file is not already in place, copy it to the
appropriate subfolder

                if my checkForFile(eachSubfolder, eachName) is false then
                    my copyFile(eachSubfolder, eachFile, eachName)
                end if

                -- Record the new file location

                set eachNewTarget to sharedPath & eachSubfolder & ":" &
eachName

                -- Replace the targets (keeping the times)

                if eachType is not "MIDI File" then
                    set currentStart to start time of eachCue
                    set currentEnd to end time of eachCue
                end if

                set file target of eachCue to eachNewTarget

                if eachType is not "MIDI File" then
                    set start time of eachCue to currentStart
                    set end time of eachCue to currentEnd
                end if

            end try

        end if

    end repeat

    display dialog "Done." with title dialogTitle with icon 1 buttons {"OK"}
default button "OK" giving up after 5

end tell

-- Subroutines

(* === FILE WRANGLING === *)

on checkForFile(theSuffix, theName) -- [Shared subroutine]
    tell application "System Events"
        return exists file (sharedPath & theSuffix & ":" & theName)
```

```
    end tell
end checkForFile

on checkForFolder(theSuffix) -- [Shared subroutine]
    tell application "System Events"
        return exists folder (sharedPath & theSuffix)
    end tell
end checkForFolder

on makeFolder(theFolder) -- [Shared subroutine]
    tell application "Finder"
        make new folder at sharedPath with properties {name:theFolder}
    end tell
end makeFolder

on copyFile(theSuffix, theFile, theName) -- [Shared subroutine]
    do shell script "cp -p " & quoted form of POSIX path of theFile & " " &
quoted form of POSIX path of (sharedPath & theSuffix & ":" & theName)
end copyFile
```

Scan for new files

This script will attempt to recreate the folder structure of a user-specified folder as Group Cues in a user-specified cue list, adding any audio/video files as Audio/Video Cues – unless they already exist. It will fail if the names of subfolders are not unique, or if cues with those names already exist in the wrong place.

```
set userWatchedFolderIsNextToWorkspace to true -- Change this to false if
your watched folder isn't next to the workspace
set userWatchedFolder to "watched" -- Set the name of the watched folder (or
change this to a full POSIX path if you change the above to false)
set userWatchedCuelist to "Watched" -- Set the name of the cue list for
automatically-generated cues
set userFlagNewCues to true -- Flag any automatically-generated cues?

-- ###FIXME### Should audioFileTypes include "public.ulaw-audio"?
-- ###FIXME### Is videoFileTypes a sufficiently exhaustive list? Are any of
the file types not supported by QLab?

-- Declarations

global dialogTitle
set dialogTitle to "Scan for files"

set audioFileTypes to {"com.apple.coreaudio-format", "com.apple.m4a-audio",
"com.microsoft.waveform-audio", "public.aifc-audio", "public.aiff-audio", ¬
    "public.audio", "public.mp3", "public.mpeg-4-audio"} -- This list
deliberately excludes "com.apple.protected-mpeg-4-audio" to protect against
(* old DRM-restricted iTunes files *)
set videoFileTypes to {"com.adobe.photoshop-image", "com.apple.icns",
```

```
"com.apple.macpaint-image", "com.apple.pict", "com.apple.quicktime-image", ¬
    "com.apple.quicktime-movie", "public.3gpp", "public.3gpp2",
"public.avi", "public.camera-raw-image", "public.image", "public.jpeg",
"public.jpeg-2000", ¬
    "public.movie", "public.mpeg", "public.mpeg-4", "public.png",
"public.tiff", "public.video", "public.xbitmap-image"}
set midiFileTypes to {"public.midi-audio"}

(* cf:
https://developer.apple.com/library/content/documentation/Miscellaneous/Refe
rence/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html *)

set theFileTypes to audioFileTypes & videoFileTypes
set cuesAdded to 0

-- Main routine

tell application id "com.figure53.QLab.4" to tell front workspace

    -- Record current selection & cue list to return to if no cues added

    set startingSelection to selected
    set startingCueList to current cue list

    -- Locate the watched folder

    if userWatchedFolderIsNextToWorkspace then

        -- Establish the path to the current workspace

        set workspacePath to path
        if workspacePath is missing value then
            display dialog "The current workspace has not yet been saved
anywhere." with title dialogTitle with icon 0 ¬
                buttons {"OK"} default button "OK" giving up after 5
            return
        end if

        -- Get the path to the watched folder

        tell application "System Events"
            set watchedFolder to path of container of file workspacePath &
userWatchedFolder
        end tell

    else

        set watchedFolder to userWatchedFolder

    end if
```

```
    -- Check watched folder exists

    tell application "System Events" to set folderExists to exists folder
watchedFolder

    if folderExists is false then
        display dialog "The watched folder \"" & POSIX path of watchedFolder
& "\" does not exist." with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK"
        return
    end if

    -- Check watched cue list exists

    try
        set watchedCuelist to first cue list whose q name is
userWatchedCuelist
    on error
        display dialog "The destination cue list \"" & userWatchedCuelist &
"\" does not exist." with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK"
        return
    end try

    -- Check watched cue list isn't a cart

    if q type of watchedCuelist is "Cart" then
        display dialog "The destination cue list \"" & userWatchedCuelist &
"\" is a cart, so no Group Cues can be made…" with title dialogTitle with
icon 0 ¬
            buttons {"OK"} default button "OK"
        return
    end if

    -- Replicate file structure

    set theFolders to {POSIX path of watchedFolder}
    set countFolders to count theFolders
    set parentFolders to {}
    set i to 0

    set currentTIDs to AppleScript's text item delimiters
    set AppleScript's text item delimiters to "/"

    repeat until i = countFolders

        set eachFolder to item (i + 1) of theFolders

        -- Record the parent of each folder processed

        if i is 0 then
```

```
            set end of parentFolders to last text item of eachFolder
        else
            set end of parentFolders to text item -2 of eachFolder
        end if

        -- Make a list of all Group Cues in the watched cue list

        set existingGroups to cues of watchedCuelist whose q type is "Group"
        set countExistingGroups to count existingGroups
        set j to 0

        repeat until j = countExistingGroups
            try
                set existingGroups to existingGroups & (cues of item (j + 1)
of existingGroups whose q type is "Group")
            end try
            set j to j + 1
            set countExistingGroups to count existingGroups
        end repeat

        -- Find first Group Cue whose name matches

        set makeNextCueIn to watchedCuelist -- If there is no match, new
cues will be added directly to the cue list
        repeat with eachGroup in existingGroups
            if q name of eachGroup is item (i + 1) of parentFolders then
                set makeNextCueIn to eachGroup
                exit repeat
            end if
        end repeat

        -- Move the selection to set where the next cue will be made

        try
            set selected to last item of (cues of makeNextCueIn)
        on error
            set current cue list to watchedCuelist
        end try

        -- Make Group Cues if needed

        if i is 0 then
            set currentGroup to watchedCuelist
        else
            set groupName to last text item of eachFolder
            set currentGroup to false
            repeat with eachGroup in existingGroups
                if q name of eachGroup is groupName then
                    set currentGroup to eachGroup
                    exit repeat
                end if
```

```
            end repeat
            if currentGroup is false then
                make type "Group"
                set cuesAdded to cuesAdded + 1
                set newCue to last item of (selected as list)
                if userFlagNewCues then set flagged of newCue to true
                set q name of newCue to groupName
                set currentGroup to newCue
            end if
        end if

        -- Make a list of files already used in the current Group Cue

        set existingTargets to file target of cues of currentGroup whose
broken is false and q type is "Audio" or q type is "Video"
        set usedFiles to {}
        repeat with eachTarget in existingTargets
            set end of usedFiles to POSIX path of eachTarget
        end repeat

        -- Get files of folder to be processed

        tell application "System Events" to set theFiles to POSIX path of
disk items of folder eachFolder whose visible is true

        -- Process them: if folder, add to list for processing; if file,
make a cue if needed

        repeat with eachFile in theFiles

            try -- This detects folders
                tell application "System Events"
                    set eachType to type identifier of file eachFile
                end tell
                if eachType is in theFileTypes and eachFile is not in
usedFiles then
                    if eachType is in audioFileTypes then
                        make type "Audio"
                    else
                        make type "Video"
                    end if
                    set cuesAdded to cuesAdded + 1
                    set newCue to last item of (selected as list)
                    if userFlagNewCues then set flagged of newCue to true
                    set file target of newCue to eachFile
                    if parent of newCue is not currentGroup then -- Will
need to move first cue made if Group Cue was empty
                        set newCueID to uniqueID of newCue
                        move cue id newCueID of parent of newCue to end of
currentGroup
                        set selected to newCue
```

```
                    end if
                end if
            on error
                set theFolders to theFolders & eachFile
            end try

        end repeat

        set i to i + 1
        set countFolders to count theFolders

    end repeat

    set AppleScript's text item delimiters to currentTIDs

    -- Report how many cues added

    if cuesAdded is 0 then
        set theMessage to "No cues were added."
        delay 1
        set selected to startingSelection
        set current cue list to startingCueList
    else if cuesAdded is 1 then
        set theMessage to "1 cue was added."
    else
        set theMessage to (cuesAdded & " cues were added.") as text
    end if

    display dialog theMessage with title dialogTitle with icon 1 buttons
{"OK"} default button "OK" giving up after 5

end tell
```

Transport

Nudge

Nudge selected cue(s) forward 5s (without changing their running state):

```
-- Only works properly when run as a separate process!
-- NB: the OSC command will NOT WORK if the workspace has a Passcode

set userNudge to 5

tell application id "com.figure53.QLab.4" to tell front workspace
    repeat with eachCue in (selected as list)
        if q type of eachCue is not "Script" then -- Protect the script from
running on itself
            try
```

```
                if running of eachCue is true then
                    pause eachCue
                    set startFlag to true
                else
                    set startFlag to false
                end if

                (* -- AppleScript method, which inadvertently resets Audio
Cues to their programmed levels
                set currentTime to action elapsed of eachCue
                load eachCue time currentTime + userNudge
                *)

                set currentTime to ((action elapsed of eachCue) - (pre wait
of eachCue)) -- loadActionAt method adds pre wait back to time argument!
                set eachID to uniqueID of eachCue
                tell me to do shell script "echo '/cue_id/" & eachID &
"/loadActionAt " & currentTime + userNudge & "' | nc -u -w 0 localhost
53535"

                if startFlag is true then
                    start eachCue
                end if

            end try
        end if
    end repeat
end tell
```

Toggle pause of all except selected cue(s)

Toggle pause of all except selected cue(s) (the script will include children of selected cues, but not their children: grandchildren will be paused):

```
tell front workspace

    -- Make a list of selected cues, including one layer of children

    set selectedIDs to {}
    repeat with eachCue in (selected as list)
        if q type of eachCue is not "Group" then
            set end of selectedIDs to uniqueID of eachCue
        else
            set selectedIDs to selectedIDs & uniqueID of cues of eachCue
        end if
    end repeat

    -- Make a list of active cues that aren't Group Cues and aren't selected
```

```
    set alreadyPaused to false
    set activeIDs to {}
    set activeCues to (active cues as list)
    repeat with eachCue in activeCues
        if q type of eachCue is not "Group" then
            set eachID to uniqueID of eachCue
            if eachID is not in selectedIDs then
                set end of activeIDs to eachID
            end if
        end if
    end repeat

    -- If any of the "active cues" are paused, start them and set a flag not
to pause any

    repeat with eachID in activeIDs
        if paused of cue id eachID is true then
            start cue id eachID
            set alreadyPaused to true
        end if
    end repeat

    -- If none of the "active cues" were paused, pause them all

    if alreadyPaused is false then
        repeat with eachID in activeIDs
            pause cue id eachID
        end repeat
    end if

end tell
```

Loading

Load nest of cues to waveform cursor

###EXPERIMENTAL###: this script was developed for a specific show, so may not be generally useful. If you click in the waveform of an Audio Cue and run the script it will attempt to load the entire sequence of nested cues you are in to that point in time: useful for choosing a cue point visually. It assumes the sequence hierarchy is entirely “Timeline” Group Cues, although it can cope with the parent of the selected cue being a “Start first child and go to next cue” Group Cue. The load time calculated is copied to the Clipboard for you to paste in next time.

```
tell front workspace

    if q type of current cue list is "Cart" then return -- This will stop
the script if we're in a cart, as it doesn't make sense to continue!

    try -- This protects against no selection (can't get last item of
```

```
(selected as list))

        set selectedCue to last item of (selected as list)

        -- Get cursor position

        set loadTime to (pre wait of selectedCue) + (percent action elapsed
of selectedCue) * (duration of selectedCue)
        -- ###FIXME### As of 4.4.1, "action elapsed" reports differently
between clicking in waveform and loading to time when rate ≠ 1

        set parentCue to parent of selectedCue

        -- If the cue is in a "Start first child and go to next cue" Group
Cue, all the cues before it will need to be loaded too;
        (* this won't detect if the cues before selectedCue won't in fact
follow on into it! *)

        if mode of parentCue is fire_first_go_to_next_cue then
            repeat with eachChild in cues of parentCue
                if contents of eachChild is selectedCue then exit repeat
                set loadTime to loadTime + (pre wait of eachChild)
                set eachContinueMode to continue mode of eachChild
                if eachContinueMode is auto_continue then
                    set loadTime to loadTime + (post wait of eachChild)
                else if eachContinueMode is auto_follow then
                    set loadTime to loadTime + (duration of eachChild)
                end if
            end repeat
        end if

        -- Go up the hierarchy until you find a Group Cue directly in a cue
list

        repeat until q type of parent of parentCue is "Cue List" -- This
will throw an error if the selected cue is directly in a cue list
            set loadTime to loadTime + (pre wait of parentCue)
            set parentCue to parent of parentCue
        end repeat

        set loadTime to loadTime + (pre wait of parentCue) -- Also include
the top level cue's Pre Wait

        -- Load the cue

        stop selectedCue
        set selected to parentCue
        load parentCue time loadTime

        -- Copy the load time to the Clipboard
```

```
        set the clipboard to loadTime as text

    end try

end tell
```

Load to time

Load selected cue(s) to a time you enter (this enhances the built-in functionality as it can act on more than one cue at a time; NB: running cues can't be loaded to time):

```
-- Declarations

global dialogTitle
set dialogTitle to "Load to time"

-- Check the Clipboard for a previous time

try
    set clipboardContents to the clipboard as text -- The time requested
previously will have been copied to the Clipboard, and may still be on there
on error
    set clipboardContents to ""
end try

if (count paragraphs of clipboardContents) > 1 or (count words of
clipboardContents) > 2 or ¬
    ((count words of clipboardContents) > 1 and clipboardContents does not
contain ":") then -- Slight protection against spurious Clipboard contents
    set clipboardContents to ""
end if

-- Prompt to get the time

set promptText to "Load selected cues to this time (seconds or
minutes:seconds):" & return & return ¬
    & "(You can enter a negative value to specify a time remaining.)"
set {theTime, theOption, inputText} to
enterATimeWithIconWithExtraOptionButton(promptText, clipboardContents,
"Start the cues too", true, true)

-- Copy the input text to the Clipboard

set the clipboard to inputText as text

-- Load (and start) the cues

tell front workspace
    if theTime < 0 then
        repeat with eachCue in (selected as list)
```

```
            load eachCue time ((pre wait of eachCue) + (duration of eachCue)
+ theTime)
        end repeat
    else
        load selected time theTime
    end if
    if theOption is "Start the cues too" then
        start selected
    end if
end tell

-- Subroutines

(* === INPUT === *)

on enterATimeWithIconWithExtraOptionButton(thePrompt, defaultAnswer,
extraOptionButton, clearDefaultAnswerAfterFirst, negativeAllowed) -- [Shared
subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
of (display dialog thePrompt with title dialogTitle with icon 1 ¬
                default answer defaultAnswer buttons (extraOptionButton as
list) & {"Cancel", "OK"} default button "OK" cancel button "Cancel")
            if clearDefaultAnswerAfterFirst is true then
                set defaultAnswer to ""
            end if
            try
                set theAnswer to theQuestion as number
                if negativeAllowed is false then
                    if theAnswer < 0 then
                        set theQuestion to ""
                    end if
                end if
            on error
                if theQuestion contains ":" then
                    if theQuestion begins with "-" then
                        if negativeAllowed is false then
                            set theAnswer to false
                        else
                            set theAnswer to -(my makeSecondsFromM_S(text 2
thru end of theQuestion))
                        end if
                    else
                        set theAnswer to my makeSecondsFromM_S(theQuestion)
                    end if
                    if theAnswer is false then
                        set theQuestion to ""
                    end if
                else
```

```
                    set theQuestion to ""
                end if
            end try
        end repeat
        return {theAnswer, theButton, theQuestion}
    end tell
end enterATimeWithIconWithExtraOptionButton

(* === TIME === *)

on makeSecondsFromM_S(howLong) -- [Shared subroutine]
    try
        set currentTIDs to AppleScript's text item delimiters
        set AppleScript's text item delimiters to ":"
        set theMinutes to first text item of howLong
        set theSeconds to rest of text items of howLong as text
        set AppleScript's text item delimiters to currentTIDs
        return theMinutes * 60 + theSeconds
    on error
        return false
    end try
end makeSecondsFromM_S
```

Jump into a string of cues

Jump into a string of cues; the script rather assumes that you are working in such a way that every cue that is triggered by a manual GO is a Group Cue (either “Timeline” or “Start first child and go to next cue”). In addition to Group Cues, it will also process any selected Memo Cues – but it will ignore all other cue types. For best results, select the first Group Cue that should still be playing and all intervening cues up to and including the last Group Cue you wish to jump into, but not any of its children. (Doesn't fire in a cart!)

```
-- Declarations

global dialogTitle
set dialogTitle to "Jump into a string of cues"

-- Main routine

tell front workspace

    -- Check we're not in a cart

    if q type of current cue list is "Cart" then return

    -- Check more than one cue selected

    try
        set selectedCues to items 1 thru -2 of (selected as list)
    on error
```

```
        display dialog "You need to select more than one cue!" with title
dialogTitle with icon 0 buttons {"OK"} default button "OK" giving up after 5
        return
    end try

    -- Check the Clipboard for a previous time

    try
        set clipboardContents to the clipboard as text -- The time requested
previously will have been copied to the Clipboard, and may still be on there
    on error
        set clipboardContents to ""
    end try

    if (count paragraphs of clipboardContents) > 1 or (count words of
clipboardContents) > 2 or ¬
        ((count words of clipboardContents) > 1 and clipboardContents does
not contain ":") then -- Slight protection against spurious Clipboard
contents
        set clipboardContents to ""
    end if

    -- Prompt to get the time

    set promptText to "This script will load the last selected cue to the
time you enter below and attempt to load any other Group Cues selected " & ¬
        "so that any fades will effectively have completed when you start
the selected cues." & return & return & ¬
        "THIS IS NOT GUARANTEED TO WORK!" & return & return & "Enter the
load time (seconds or minutes:seconds):"
    set {theTime, unusedVariable, inputText} to my
enterATimeWithIconWithExtraOptionButton(promptText, clipboardContents, {},
true, false)

    -- Copy the input text to the Clipboard

    set the clipboard to inputText as text

    -- Clean out cues that won't be processed, and prepare a list for
checking to see if remaining selection contains its own children…

    set selectedCuesClean to {}
    set childrenIDs to {}
    repeat with eachCue in selectedCues
        if q type of eachCue is in {"Group", "Memo"} then
            set end of selectedCuesClean to eachCue
            try -- This will fail silently for Memo Cues
                set childrenIDs to childrenIDs & uniqueID of cues of eachCue
            end try
        end if
    end repeat
```

```
    -- Exit if no cues left to process

    if (count selectedCuesClean) is 0 then
        display dialog "Not enough Group or Memo Cues selected to proceed."
with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK" giving up after 5
        return
    end if

    -- Work out the total time to which to load, and temporarily set each
Group/Memo Cue to auto-continue

    set longestGroupTime to 0
    set summedWaits to 0
    set processedCues to {}
    set continueModes to {}

    repeat with eachCue in selectedCuesClean

        try

            set groupTime to 0

            try
                set eachMode to mode of eachCue
            on error -- Memo Cue
                set eachMode to ""
            end try

            -- Skip cues that are children of selected Group Cues

            if uniqueID of eachCue is in childrenIDs then
                error -- Skip any selected children of selected Group Cues
so as not to process them twice
            end if

            (* The total time of a "Timeline" Group Cue – the time to which
to load it to "complete" -
            is the sum of the Pre Wait and Duration of the Fade Cue that
will take longest to complete *)

            if eachMode is timeline then
                set fadeCues to (cues of eachCue whose q type is "Fade")
                set longestChildFadeTime to 0
                repeat with eachChild in fadeCues
                    set eachChildFadeTime to (pre wait of eachChild) +
(duration of eachChild)
                    if eachChildFadeTime > longestChildFadeTime then
                        set longestChildFadeTime to eachChildFadeTime
                    end if
```

```
                end repeat
                set groupTime to longestChildFadeTime
            end if

            (* The total time of a "Start first child and go to next cue"
Group Cue — the time to which to load it to "complete" -
            is the sum of all the Pre Waits and Post Waits of cues that
continue, plus the Duration of the longest Fade Cue *)

            if eachMode is fire_first_go_to_next_cue then
                set longestChildFadeTime to 0
                repeat with eachChild in cues of eachCue
                    set groupTime to groupTime + (pre wait of eachChild)
                    set eachContinueMode to continue mode of eachChild
                    if eachContinueMode is auto_continue then
                        set groupTime to groupTime + (post wait of
eachChild)
                        if q type of eachChild is "Fade" then
                            set eachChildFadeTime to duration of eachChild
                            if eachChildFadeTime > longestChildFadeTime then
                                set longestChildFadeTime to
eachChildFadeTime
                            end if
                        end if
                    else if eachContinueMode is auto_follow then
                        set groupTime to groupTime + (duration of eachChild)
                    else
                        exit repeat -- No point looking at children after
this as they aren't part of the sequence; this child's Pre Wait has been
counted
                    end if
                end repeat
                set groupTime to groupTime + longestChildFadeTime
            end if

            -- Since the Group Cues are being set to auto-continue, loading
to the longest of their total times will load them all to "completion"

            if groupTime > longestGroupTime then
                set longestGroupTime to groupTime
            end if

            (* If any of the Group/Memo Cues have non-zero Pre or Post Waits
then these will effectively extend the time to which we have to load,
            so these are summed too *)

            set end of processedCues to eachCue
            set end of continueModes to continue mode of eachCue
            set continue mode of eachCue to auto_continue
            set summedWaits to summedWaits + (pre wait of eachCue)
            if contents of eachCue is not last item of selectedCuesClean
```

```
then
                (* Don't include the penultimate cue's Post wait in the sum
as it will be set temporarily to 0 *)
                set summedWaits to summedWaits + (post wait of eachCue)
            end if

        end try

    end repeat

    -- Temporarily change Post Wait of penultimate Group/Memo Cue in
selection so that when the string is loaded all other cues will "complete"

    set lastPost to post wait of last item of selectedCuesClean
    set post wait of last item of selectedCuesClean to longestGroupTime +
summedWaits

    -- Load the cue string and prompt to start it

    load first item of selectedCuesClean time longestGroupTime + summedWaits
+ theTime -- This includes the load to time specified by the user in the
dialog

    try -- This try means that the rest of the script will complete even if
the user cancels
        display dialog "Ready to go?" with title dialogTitle with icon 1
buttons {"Cancel", "OK"} default button "OK" cancel button "Cancel"
        start first item of selectedCuesClean
    end try

    -- Reset the cues

    repeat with i from 1 to count processedCues
        set continue mode of item i of processedCues to item i of
continueModes
    end repeat
    set post wait of last item of selectedCuesClean to lastPost

end tell

-- Subroutines

(* === INPUT === *)

on enterATimeWithIconWithExtraOptionButton(thePrompt, defaultAnswer,
extraOptionButton, clearDefaultAnswerAfterFirst, negativeAllowed) -- [Shared
subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
```

```
of (display dialog thePrompt with title dialogTitle with icon 1 ¬
                default answer defaultAnswer buttons (extraOptionButton as
list) & {"Cancel", "OK"} default button "OK" cancel button "Cancel")
            if clearDefaultAnswerAfterFirst is true then
                set defaultAnswer to ""
            end if
            try
                set theAnswer to theQuestion as number
                if negativeAllowed is false then
                    if theAnswer < 0 then
                        set theQuestion to ""
                    end if
                end if
            on error
                if theQuestion contains ":" then
                    if theQuestion begins with "-" then
                        if negativeAllowed is false then
                            set theAnswer to false
                        else
                            set theAnswer to -(my makeSecondsFromM_S(text 2
thru end of theQuestion))
                        end if
                    else
                        set theAnswer to my makeSecondsFromM_S(theQuestion)
                    end if
                    if theAnswer is false then
                        set theQuestion to ""
                    end if
                else
                    set theQuestion to ""
                end if
            end try
        end repeat
        return {theAnswer, theButton, theQuestion}
    end tell
end enterATimeWithIconWithExtraOptionButton

(* === TIME === *)

on makeSecondsFromM_S(howLong) -- [Shared subroutine]
    try
        set currentTIDs to AppleScript's text item delimiters
        set AppleScript's text item delimiters to ":"
        set theMinutes to first text item of howLong
        set theSeconds to rest of text items of howLong as text
        set AppleScript's text item delimiters to currentTIDs
        return theMinutes * 60 + theSeconds
    on error
        return false
    end try
end makeSecondsFromM_S
```

Navigation

Mark/Jump

Mark selected cue, or jump to previously marked cue (the script uses a property to store information about the marked cue: if there is no selection, the script will jump to the marked cue; if no cue has been marked, the script will mark the selected cue; if there is a selection and a cue has previously been marked, the script will ask what to do next):

```
-- This script can not be run as a separate process as this creates a new
instance each time, resetting the property used to store the marked cue

-- Declarations

global dialogTitle
set dialogTitle to "Mark/Jump"
property storedCue : false

-- Main routine

tell front workspace
    set selectedCues to (selected as list)
    if (count selectedCues) is 0 then -- There is no selected cue: we are
jumping
        if storedCue is not false then
            my jumpToCue()
        end if
    else
        if storedCue is false then -- There is no stored cue: we are marking
            my markCue(last item of selectedCues)
        else -- There is a stored cue, but we'll check what is required
            set theButton to button returned of (display dialog "Jump to
stored cue?" with title dialogTitle with icon 1 ¬
                buttons {"Mark", "OK"} default button "OK")
            if theButton is "OK" then -- We are jumping
                my jumpToCue()
            else -- We are marking
                my markCue(last item of selectedCues)
            end if
        end if
    end if
end tell

-- Subroutines

(* === PROCESSING === *)

on markCue(cueToMark)
    tell front workspace
```

```
        set storedCue to cueToMark
    end tell
end markCue

on jumpToCue()
    tell front workspace
        try
            set selected to storedCue -- This will throw an error if the cue
has been deleted
        on error
            set storedCue to false -- Clear the stored cue if it wasn't
found
        end try
    end tell
end jumpToCue
```

Next broken cue

Go to next broken cue (can be adjusted to search for other properties – although for the case “loaded is true”, the script inevitably finds itself, and for the case “running is true”, the script finds itself and its parent!):

```
tell front workspace

    set foundCues to uniqueID of cues whose q type is not "Cue List" and q
type is not "Cart" and ¬
        broken is true -- Change this line to search for different
properties, eg: q type is "Audio"

    set foundCuesRef to a reference to foundCues
    set countFoundCues to count foundCuesRef

    if countFoundCues is 0 then -- No cues match the search
        return
    end if

    -- Find where we are in the list of all cues

    try

        set currentCue to uniqueID of last item of (selected as list) --
This will throw an error if there is no selection

        if currentCue is in foundCuesRef then

            set currentIndex to null -- Bypass iterative searches if
selected cue is in "found" cues

        else
```

```
            set allCues to uniqueID of cues
            set allCuesRef to a reference to allCues
            set countCues to count allCuesRef

            repeat with i from 1 to countCues
                if item i of allCuesRef is currentCue then
                    set currentIndex to i -- Position of selected cue
                    exit repeat
                end if
            end repeat

        end if

    on error

        set currentIndex to 0 -- No selection; start search at beginning

        set allCues to uniqueID of cues
        set allCuesRef to a reference to allCues
        set countCues to count allCuesRef

    end try

    -- Find the next "found" cue in the list of all cues

    if currentIndex is null then

        repeat with i from 1 to countFoundCues
            if item i of foundCuesRef is currentCue then
                set foundID to item (i mod countFoundCues + 1) of
foundCuesRef -- The mod makes the search wrap around
                exit repeat
            end if
        end repeat

    else

        set foundID to null

        repeat with i from (currentIndex mod countCues + 1) to countCues --
The mod protects against selected cue being the last in the workspace
            if item i of allCuesRef is in foundCuesRef then
                set foundID to item i of allCuesRef
                exit repeat
            end if
        end repeat

        if foundID is null then -- Need to wrap around as no cue found
            repeat with i from 1 to currentIndex
                if item i of allCuesRef is in foundCuesRef then
                    set foundID to item i of allCuesRef
```

```
                    exit repeat
                end if
            end repeat
        end if

    end if

    -- Move the selection

    set selected to cue id foundID

end tell
```

Next cue which shares Cue Target

Go to next cue with the same Cue Target as the selected cue – or which targets the selected cue itself, if it doesn't have a Cue Target:

```
tell front workspace

    try -- This protects against no selection (can't get last item of
(selected as list))

        set selectedCue to last item of (selected as list)

        set targetCue to cue target of selectedCue
        if targetCue is missing value then
            set targetCue to selectedCue
        end if

        set foundCues to (uniqueID of cues whose cue target is targetCue) &
uniqueID of targetCue -- Include the shared Cue Target itself

        set foundCuesRef to a reference to foundCues
        set countFoundCues to count foundCuesRef

        -- Find where we are in the list of all cues

        set currentCue to uniqueID of selectedCue

        -- Find the next "found" cue in the list of all cues

        repeat with i from 1 to countFoundCues
            if item i of foundCuesRef is currentCue then
                set foundID to item (i mod countFoundCues + 1) of
foundCuesRef -- The mod makes the search wrap around
                exit repeat
            end if
        end repeat
```

```
        -- Move the selection

        set selected to cue id foundID

    end try

end tell
```

Navigate cues with shared Cue Target

Present a popup list of cues with the same Cue Target as the selected cue – or which target the selected cue itself, if it doesn't have a Cue Target:

```
set userPreselectNextCue to true -- Set this to false to preselect the
selected cue in the popup list

-- Declarations

global dialogTitle
set dialogTitle to "Navigate cues with shared Cue Target"

-- Main routine

tell front workspace

    try -- This protects against no selection (can't get last item of
(selected as list))

        set selectedCue to last item of (selected as list)

        set targetCue to cue target of selectedCue
        if targetCue is missing value then
            set targetCue to selectedCue
        end if

        set threadCues to {targetCue} & (cues whose cue target is targetCue)
-- Only form of this query that doesn't throw an error (brackets essential)!
        set threadCount to count threadCues
        if threadCount is 1 then -- Short thread!
            return
        end if

        set threadNames to {q list name of targetCue} & q list name of (cues
whose cue target is targetCue)
        set threadTypes to {q type of targetCue} & q type of (cues whose cue
target is targetCue)
        set threadNumbers to {q number of targetCue} & q number of (cues
whose cue target is targetCue)

        -- Make the popup list
```

```
        set threadList to {}
        set cueListDetector to cue id "[root group of cue lists]" -- Cue
lists have this as their parent
        repeat with i from 1 to threadCount
            set eachCue to item i of threadCues
            set eachName to item i of threadNames
            set eachType to item i of threadTypes
            set eachNumber to item i of threadNumbers
            if eachNumber is "" then
                set eachParent to parent of eachCue
                if parent of eachParent is not cueListDetector then --
eachCue's parent is not a cue list
                    repeat until parent of eachParent's parent is
cueListDetector -- Go up the hierarchy until you find a Group Cue directly
in a cue list
                        set eachParent to parent of eachParent
                    end repeat
                    set eachNumber to q number of eachParent
                    if eachNumber is not "" then
                        set eachNumber to "{" & eachNumber & "}" --
"Inherited" Cue Number from enclosing group(s)
                    end if
                end if
            end if
            set end of threadList to (eachNumber & tab & eachType & tab &
eachName) -- ###FIXME### This can be ugly…
        end repeat

        -- Display the list, including pre-selecting the cue after the
selected cue (or the selected cue, depending on value of
userPreselectNextCue)

        repeat with i from 1 to threadCount
            if selectedCue is item i of threadCues then
                if userPreselectNextCue is true then
                    set preSelect to (i mod threadCount) + 1
                else
                    set preSelect to i
                end if
                exit repeat
            end if
        end repeat

        set chosenCue to my pickFromListCustomDefault(threadList, "Please
choose the cue to jump to:", preSelect)

        -- Convert the answer to a cue

        repeat with i from 1 to threadCount
            if chosenCue is item i of threadList then
```

```
                set jumpToCue to item i of threadCues
                exit repeat
            end if
        end repeat

        -- Move the selection

        set selected to jumpToCue

    end try

end tell

-- Subroutines

(* === INPUT === *)

on pickFromListCustomDefault(theChoice, thePrompt, theDefault) -- [Shared
subroutine]
    tell application id "com.figure53.QLab.4"
        choose from list theChoice with prompt thePrompt with title
dialogTitle default items item theDefault of theChoice
        if result is not false then
            return item 1 of result
        else
            error number -128
        end if
    end tell
end pickFromListCustomDefault
```

View

Switch cue lists

Switch to the cue list called “Main Cue List”:

```
set userCueList to "Main Cue List" -- Use this to specify the name of the
cue list

tell front workspace
    set current cue list to first cue list whose q name is userCueList
end tell
```

Switch cue lists without losing playhead

Switch to the cue list called “Sub Cue List”; the script will attempt to navigate to the cue that would be fired by the next GO targeting this list (useful for secondary cue lists that are fired by an

independent GO button or hotkey):

```
set userCueList to "Sub Cue List" -- Use this to specify the name of the cue list

tell front workspace
    set current cue list to first cue list whose q name is userCueList
    try
        set selected to playback position of current cue list
    end try
end tell
```

Not hotkeys

Macros for one-off processes (eg: making a soundcheck, reporting, etc).

Batch adjusting

Batch modify OSC cues for localisation issues

Localise imported OSC cues if you use “,” rather than “.” for the decimal point:

```
set userSearchFor to "." -- Number separator to be replaced
set userReplaceWith to "," -- Replacement value

tell application id "com.figure53.QLab.4" to tell front workspace
    set vulnerableCues to cues whose q type is "Network" and custom message contains userSearchFor
    set currentTIDs to AppleScript's text item delimiters
    repeat with eachCue in vulnerableCues
        set AppleScript's text item delimiters to userSearchFor
        set eachMessage to custom message of eachCue
        set eachList to text items of eachMessage
        set AppleScript's text item delimiters to userReplaceWith
        set custom message of eachCue to eachList as text
    end repeat
    set AppleScript's text item delimiters to currentTIDs
end tell
```

Set every cue to auto-load

Set every cue in the workspace to auto-load:

```
/cue/*/autoLoad 1
```

Making

Make a soundcheck sequence

Create a Group Cue containing a sequence of soundcheck cues, made from an Audio Cue you choose:

```
-- Best run as a separate process so it can be happening in the background

set userSoundcheckList to "Soundcheck" -- Use this to set the name of the cue list in which to search for the root Audio Cue
set userMinVolume to -120 -- Set what level you mean by "faded out" (you can adjust this to match the workspace "Min Volume Limit" if necessary)

set userVerboseMode to true -- Set this to false to use the next 4 user settings below without any dialogs
set userNumberOfOutputsToCheck to 32 -- Set your preferred option for how many outputs to check – confirmed by dialog
set userCrossfadeDuration to 1 -- Set your preferred option for how long the crossfades should be – confirmed by dialog
set userStartSequenceWithFadeIn to true -- Set your preferred option for fading in at the start – confirmed by dialog
set userAutomaticFollowOnTime to "" -- Set the time spent at each output (or "" for no follow-ons) – confirmed by dialog

set userDefaultCueNumberForSoundcheckCue to "999" -- Set the Cue Number for the soundcheck cue (or "" for workspace default)

set userStartCueName to "Start soundcheck" -- Set the name for the first cue
set userFadeInCueName to "Fade in output " -- Set the name for the fade-in cue, if used
set userMoveCuesBaseName to "Move to output " -- Set the base name for the move cues
set userMoveCuesBaseNotes to "You are listening to output " -- Set the base Notes for the move cues
set userFollowOnCueNames to {"     Automatic follow-on…", "     …Automatic follow-on"} -- Set the names for the cues used to create the follow-ons, if used
set userStopCueName to "Stop soundcheck" -- Set the name for the final cue

-- Explanations

set theExplanation to "This script will create a Group Cue containing a sequence of soundcheck cues made from an Audio Cue you choose from the \"" & ¬
    userSoundcheckList & "\" cue list (the Audio Cue needs to be directly in the cue list, not inside a Group Cue). " & ¬
    "The soundcheck will step through one output at a time via a sequence of default crossfades.
```

```
Before running the script you should configure the Audio Cue to play at the
desired levels out of all the outputs you wish to check, " & ¬
    "as these levels will be used in the fades. The script will not adjust
crosspoints.

Several additional user settings can be adjusted by editing the script
itself."

-- Declarations

global dialogTitle
set dialogTitle to "Make a soundcheck sequence"

set qLabMaxAudioChannels to 64

-- Preamble

tell application id "com.figure53.QLab.4" to tell front workspace

    -- Display introductory dialog, if in verbose mode

    if userVerboseMode is true then
        display dialog theExplanation with title dialogTitle with icon 1
buttons {"Cancel", "OK"} default button "OK" cancel button "Cancel"
    end if

    -- Exit if userSoundcheckList doesn't exist

    try
        set soundcheckList to first cue list whose q name is
userSoundcheckList
    on error
        display dialog "The cue list \"" & userSoundcheckList & "\" can not
be found…" with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK" giving up after 5
        return
    end try

    -- Exit if userSoundcheckList is a cart

    try
        set notCaseSensitiveMatch to q name of soundcheckList
        if q type of soundcheckList is "Cart" then
            display dialog "The destination cue list \"" &
notCaseSensitiveMatch & "\" is a cart, so no Group Cues can be made…" with
title dialogTitle ¬
                with icon 0 buttons {"OK"} default button "OK"
            return
        end if
    end try
```

```
    -- Exit if there aren't any Audio Cues in userSoundcheckList

    set possibleCues to cues of soundcheckList whose q type is "Audio" and
broken is false
    set countPossibleCues to count possibleCues

    if countPossibleCues is 0 then
        display dialog "No Audio Cues found directly in the \"" &
userSoundcheckList & "\" cue list." with title dialogTitle with icon 0 ¬
            buttons {"OK"} default button "OK" giving up after 5
        return
    end if

    -- Check to see if any possible Audio Cues have Cue Numbers, and prepare
the choose from list dialog accordingly

    set possibleCueNames to q list name of cues of soundcheckList whose q
type is "Audio" and broken is false
    set possibleCueNumbers to q number of cues of soundcheckList whose q
type is "Audio" and broken is false

    set currentTIDs to AppleScript's text item delimiters
    set AppleScript's text item delimiters to "" -- Should not assume this
    if possibleCueNumbers as text is not "" then
        repeat with i from 1 to countPossibleCues
            set item i of possibleCueNames to (item i of possibleCueNumbers
& tab & item i of possibleCueNames) as text
        end repeat
    end if
    set AppleScript's text item delimiters to currentTIDs

    -- Choose which Audio Cue

    set theAudioCueName to my pickFromList(possibleCueNames, "Please choose
the Audio Cue:")

    repeat with i from 1 to countPossibleCues
        if theAudioCueName is item i of possibleCueNames then
            set theAudioCue to item i of possibleCues
            exit repeat
        end if
    end repeat

    -- Get all output levels from the Audio Cue (except Master)

    set allOutCheck to true
    set originalCueLevels to {}
    repeat with i from 1 to qLabMaxAudioChannels
        set thisOutputLevel to theAudioCue getLevel row 0 column i
        if thisOutputLevel is not userMinVolume then
            set allOutCheck to false
```

```
        end if
        set end of originalCueLevels to thisOutputLevel
    end repeat

    -- Exit if there are no levels set in the Audio Cue

    if allOutCheck is true then
        display dialog "The last selected Audio Cue has all its individual
output levels set to \"-INF\".

It makes no sense to proceed…" with title dialogTitle with icon 0 buttons
{"OK"} default button "OK"
        return
    end if

    -- Skip the dialogs if userVerboseMode is false

    if userVerboseMode is false then

        set howManyOutputs to userNumberOfOutputsToCheck
        set howLong to userCrossfadeDuration
        set fadeIn to userStartSequenceWithFadeIn
        set followOn to userAutomaticFollowOnTime

    else

        -- Prompt for how many outputs to test

        set howManyOutputs to my enterANumberWithRangeWithCustomButton("How
many outputs do you wish to check?", userNumberOfOutputsToCheck, ¬
            2, true, qLabMaxAudioChannels, true, true, {}, 2)

        -- Prompt for length of crossfades

        set howLong to my enterANumberWithRangeWithCustomButton("Enter a
duration for the crossfades (in seconds):", userCrossfadeDuration, ¬
            0, true, false, false, false, {}, 2)

        -- Prompt for whether there should be a fade in at the start

        if userStartSequenceWithFadeIn is true then
            set fadeButtons to {"Cancel", "No", "Yes"}
        else
            set fadeButtons to {"Cancel", "Yes", "No"}
        end if
        set fadeQuestion to button returned of (display dialog "Would you
like the sequence to start by fading in?" with title dialogTitle with icon 1
¬
            buttons fadeButtons default button 3 cancel button 1)
        if fadeQuestion is "Yes" then
            set fadeIn to true
```

```
        else
            set fadeIn to false
        end if

        -- Prompt for whether the sequence should automatically follow-on

        set followOnMessage to "Set the time spent at each output (in
seconds):"
        if userAutomaticFollowOnTime is "" then
            set followOnDefault to 1
            set followOnMessage to followOnMessage & return & return &
"(You'll need to click \"OK\" to enter a time, not press return.)"
            -- ###FIXME### This could be more elegant: use text returned to
test for an entry
        else
            set followOnDefault to 3
        end if

        set followOn to my
enterANumberWithRangeWithCustomButton(followOnMessage,
userAutomaticFollowOnTime, ¬
            0, true, false, false, false, "No follow-ons", followOnDefault)

        if followOn is "No follow-ons" then
            set followOn to ""
        end if

    end if

    -- The bit of the script that actually does the work starts here…

    display dialog "One moment caller…" with title dialogTitle with icon 1
buttons {"OK"} default button "OK" giving up after 1

    -- Switch to userSoundcheckList

    set current cue list to soundcheckList

    -- Make a new Group Cue for the sequence: after the Audio Cue

    set selected to theAudioCue -- This also protects against the default
behaviour of grouping selection if more than 2 selected
    make type "Group"
    set theGroupCue to last item of (selected as list)
    set mode of theGroupCue to fire_first_enter_group
    set q name of theGroupCue to userStartCueName
    if userDefaultCueNumberForSoundcheckCue is not "" then
        set q number of theGroupCue to userDefaultCueNumberForSoundcheckCue
    end if

    -- Move the Audio Cue inside the Group Cue
```

```
    set theAudioCueID to uniqueID of theAudioCue
    set theAudioCueIsIn to parent of theAudioCue
    set theGroupCueID to uniqueID of theGroupCue
    move cue id theAudioCueID of theAudioCueIsIn to end of cue id
theGroupCueID
    set selected to theAudioCue -- The Group Cue was the last selection, so
we need to select a cue inside the group before making the fades

    -- Set outputs 2 & above to -INF (do all the outputs regardless of
userNumberOfOutputsToCheck so there's no unexpected audio from higher output
numbers)

    set outputOneGang to getGang theAudioCue row 0 column 1
    if outputOneGang is not missing value then
        setGang theAudioCue row 0 column 1 gang "" -- Temporarily override
gang for output 1 (it affects setLevel on the Audio Cue, but not on any
fades)
    end if

    repeat with i from 2 to qLabMaxAudioChannels
        theAudioCue setLevel row 0 column i db userMinVolume
    end repeat

    if outputOneGang is not missing value then
        setGang theAudioCue row 0 column 1 gang outputOneGang
    end if

    -- Create fade in, if necessary

    if fadeIn is true then
        theAudioCue setLevel row 0 column 1 db userMinVolume
        set continue mode of theAudioCue to auto_continue
        make type "Fade"
        set newCue to last item of (selected as list)
        set cue target of newCue to theAudioCue
        set duration of newCue to howLong
        set newCueLevels to item 1 of originalCueLevels
        newCue setLevel row 0 column 1 db newCueLevels
        if followOn is "" then
            set q name of newCue to userFadeInCueName & "1"
        else
            my makeCrashableWait(newCue, followOn, true, ¬
                userFadeInCueName & "1", item 2 of userFollowOnCueNames,
userMoveCuesBaseName & "2", userMoveCuesBaseNotes & "1")
        end if
    else
        if followOn is "" then
            set continue mode of theAudioCue to do_not_continue
        else
            my makeCrashableWait(theAudioCue, followOn, false, ¬
```

```
                false, item 2 of userFollowOnCueNames, userMoveCuesBaseName
& "2", userMoveCuesBaseNotes & "1")
        end if
    end if

    -- Make fades

    repeat with i from 2 to howManyOutputs
        make type "Fade"
        set newCue to last item of (selected as list)
        set cue target of newCue to theAudioCue
        set duration of newCue to howLong
        set newCueLevels to item i of originalCueLevels
        newCue setLevel row 0 column i db newCueLevels
        newCue setLevel row 0 column (i - 1) db userMinVolume
        if followOn is "" then
            set q name of newCue to userMoveCuesBaseName & i
            set notes of newCue to userMoveCuesBaseNotes & (i - 1)
        else if i < howManyOutputs then
            my makeCrashableWait(newCue, followOn, true, ¬
                item 1 of userFollowOnCueNames, item 2 of
userFollowOnCueNames, userMoveCuesBaseName & (i + 1), userMoveCuesBaseNotes
& i)
        else
            my makeCrashableWait(newCue, followOn, true, ¬
                item 1 of userFollowOnCueNames, item 2 of
userFollowOnCueNames, userStopCueName, userMoveCuesBaseNotes & i)
        end if
    end repeat

    -- Make final fade out

    make type "Fade"
    set newCue to last item of (selected as list)
    set cue target of newCue to theAudioCue
    set duration of newCue to howLong
    newCue setLevel row 0 column howManyOutputs db userMinVolume
    set stop target when done of newCue to true
    if followOn is "" then
        set q name of newCue to userStopCueName
        set notes of newCue to userMoveCuesBaseNotes & howManyOutputs
    else
        set q name of newCue to item 2 of userFollowOnCueNames
    end if

    activate
    display dialog "Done." with title dialogTitle with icon 1 buttons {"OK"}
default button "OK" giving up after 60

end tell
```

```
-- Subroutines

(* === INPUT === *)

on enterANumberWithRangeWithCustomButton(thePrompt, defaultAnswer, ¬
    lowRange, acceptEqualsLowRange, highRange, acceptEqualsHighRange,
integerOnly, customButton, defaultButton) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        set theQuestion to ""
        repeat until theQuestion is not ""
            set {theQuestion, theButton} to {text returned, button returned}
of (display dialog thePrompt with title dialogTitle ¬
                default answer defaultAnswer buttons (customButton as list)
& {"Cancel", "OK"} default button defaultButton cancel button "Cancel")
            if theButton is customButton then
                set theAnswer to theButton
                exit repeat
            end if
            try
                if integerOnly is true then
                    set theAnswer to theQuestion as integer -- Detects non-
numeric strings
                    if theAnswer as text is not theQuestion then -- Detects
non-integer input
                        set theQuestion to ""
                    end if
                else
                    set theAnswer to theQuestion as number -- Detects non-
numeric strings
                end if
                if lowRange is not false then
                    if acceptEqualsLowRange is true then
                        if theAnswer < lowRange then
                            set theQuestion to ""
                        end if
                    else
                        if theAnswer ≤ lowRange then
                            set theQuestion to ""
                        end if
                    end if
                end if
                if highRange is not false then
                    if acceptEqualsHighRange is true then
                        if theAnswer > highRange then
                            set theQuestion to ""
                        end if
                    else
                        if theAnswer ≥ highRange then
                            set theQuestion to ""
                        end if
                    end if
```

```
                end if
            on error
                set theQuestion to ""
            end try
        end repeat
        return theAnswer
    end tell
end enterANumberWithRangeWithCustomButton

on pickFromList(theChoice, thePrompt) -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        choose from list theChoice with prompt thePrompt with title
dialogTitle default items item 1 of theChoice
        if result is not false then
            return item 1 of result
        else
            error number -128
        end if
    end tell
end pickFromList

(* === PROCESSING === *)

on makeCrashableWait(originalCue, waitTime, autoFollow, originalCueName,
startCueName, stopCueName, stopCueNotes)
    tell application id "com.figure53.QLab.4" to tell front workspace
        if autoFollow is false then
            set continue mode of originalCue to auto_continue
        else
            set continue mode of originalCue to auto_follow
        end if
        if originalCueName is not false then
            set q name of originalCue to originalCueName
        end if
        make type "Start"
        set newStartCue to last item of (selected as list)
        set cue target of newStartCue to current cue list
        set pre wait of newStartCue to waitTime
        set q name of newStartCue to startCueName
        make type "Stop"
        set newStopCue to last item of (selected as list)
        set cue target of newStopCue to newStartCue
        set q name of newStopCue to stopCueName
        set notes of newStopCue to stopCueNotes -- The "crashable wait"
pauses on the Stop Cue
        set continue mode of newStopCue to auto_continue
    end tell
end makeCrashableWait
```

Reporting

Log event to file

###EXPERIMENTAL###

Fire this script with a Start Cue with a 1s Post Wait – the name of the Start Cue will be logged to a text file along with the time of day:

```
-- Best run as a separate process

set userLogsFolder to false -- Default behaviour is to create log files on the Desktop; to override, enter a path to a different folder here (it needs to exist)
(* The easiest way to get the path is to select the folder in the Finder, press alt-cmd-C and paste the resulting path as a "string" above, replacing 'false' *)

set userCommonLogFile to false -- Default behaviour is to write a new log file each day and log workspaces in separate files;
(* if you want to have a single ongoing file that logs all workspaces, change this variable to the name of the file, eg: "QLab Log" *)

set userFileText to " | Logged Events | " -- Set the text to be inserted into the filename between workspace name and today's date, if in that default mode

-- Get cue that started this Script Cue and get the time

tell application id "com.figure53.QLab.4" to tell front workspace
    try
        set startCueName to q list name of item -3 of (active cues as list)
-- This bit only works if this script is inside a Group Cue!
    on error
        set startCueName to false
    end try
end tell

set logTime to do shell script "date '+%d-%m-%y %T'"

-- Declarations

global dialogTitle
set dialogTitle to "Log event to file"

-- Check userLogsFolder exists

if userLogsFolder is false then -- Default to the Desktop
    set logsFolder to path to desktop
else
```

```
    try
        tell application "System Events"
            set logsFolder to path of folder userLogsFolder -- This will
throw an error if the folder doesn't exist
        end tell
    on error
        display dialog "The folder \"" & userLogsFolder & "\" can not be
accessed; please revisit userLogsFolder variable." with title dialogTitle
with icon 0 ¬
            buttons {"OK"} default button "OK"
        return
    end try
end if

-- Get info for the log

tell application id "com.figure53.QLab.4" to tell front workspace
    set workspaceName to q number
    if startCueName is false then
        set theMessage to "**** Start Cue name not captured ****"
    else
        set theMessage to startCueName
    end if
    if userCommonLogFile is not false then
        set theMessage to workspaceName & tab & theMessage
    end if
end tell

-- Get variables for file naming

set fileDate to do shell script "date '+%y-%m-%d'"

if userCommonLogFile is false then
    set theFilePath to POSIX path of logsFolder & workspaceName &
userFileText & fileDate & ".txt"
else
    set theFilePath to POSIX path of logsFolder & userCommonLogFile & ".txt"
end if

-- Write to the log

do shell script "echo " & logTime & tab & quoted form of theMessage & " >> "
& quoted form of theFilePath
```

Make a list of media files

Make a text file on the Desktop with a list of all the media files used as targets in the current workspace:

```
-- Best run as a separate process so it can be happening in the background
```

```
-- Explanations

set theExplanation to "This script creates a text file on the Desktop with a
list of all the media files used as targets in the current workspace, " & ¬
    "removing duplicates and sorting alphabetically. It then opens the file
in TextEdit."

-- Declarations

global dialogTitle, startTime
set dialogTitle to "Make a list of media files"

-- Preamble

tell application id "com.figure53.QLab.4"

    display dialog theExplanation & return & return & "It may take a little
while to run; are you sure you wish to proceed?" with title dialogTitle with
icon 1 ¬
        buttons {"Cancel", "OK"} default button "OK" cancel button "Cancel"

    -- Prepare variables for filename

    set fileTime to my grabTimeForFilename()
    set workspaceName to q number of front workspace -- This actually gets
the name of the workspace

    my startTheClock()

    -- Extract array of File Targets from QLab, skipping duplicates

    set mediaFiles to {}
    set mediaFilesRef to a reference to mediaFiles

    tell front workspace

        -- First, the easy ones

        set validTargets to file target of cues whose broken is false and (q
type is "Audio" or q type is "Video" or q type is "MIDI File")
        set countValidTargets to count validTargets

        repeat with i from 1 to countValidTargets

            set eachTarget to item i of validTargets
            set targetFile to POSIX path of alias eachTarget -- Convert to
POSIX
            if targetFile is not in mediaFilesRef then
                set end of mediaFilesRef to targetFile
            end if
```

```
            if i mod 100 is 0 and (countValidTargets - i) > 50 then --
Countdown timer (and opportunity to escape)
                my countdownTimer(i, countValidTargets, "valid Audio, Video
or MIDI File Cues")
            end if

        end repeat

        -- Now, broken cues

        set brokenCues to cues whose broken is true and (q type is "Audio"
or q type is "Video" or q type is "MIDI File")
        set countBrokenCues to count brokenCues

        repeat with i from 1 to countBrokenCues

            set eachCue to item i of brokenCues
            set eachTarget to file target of eachCue
            if eachTarget is missing value then -- This will be returned by
cues whose targets have become invalid
                set eachName to q display name of eachCue
                set targetFile to "**** Missing file target for cue named
\"" & eachName & "\" ****"
            else
                set targetFile to POSIX path of alias eachTarget & "
[BROKEN]"
            end if
            if targetFile is not in mediaFilesRef then
                set end of mediaFilesRef to targetFile
            end if

            if i mod 100 is 0 and (countBrokenCues - i) > 50 then --
Countdown timer (and opportunity to escape)
                my countdownTimer(i, countBrokenCues, "broken Audio, Video
or MIDI File Cues")
            end if

        end repeat

    end tell

    -- Check there are some files

    if (count mediaFilesRef) is 0 then
        activate
        display dialog "No media files were found!" with title dialogTitle
with icon 0 ¬
            buttons {"OK"} default button "OK" giving up after 60
        return
    end if
```

```
end tell

-- Convert the list to text and sort it

set currentTIDs to AppleScript's text item delimiters
set AppleScript's text item delimiters to linefeed
set theText to mediaFilesRef as text
set AppleScript's text item delimiters to currentTIDs

set sortedText to sortTextIgnoringCase(theText)

-- Create a string of the full path of the text file to be created

set newFile to "" & (path to desktop) & "QLab | " & workspaceName & " |
Media files in use | " & fileTime & ".txt"

-- Make the file

makeFileFromText(newFile, sortedText)

-- Open it in TextEdit

tell application "TextEdit"
    activate
    open file newFile
    set zoomed of front window to true
end tell

finishedDialogBespoke()

-- Subroutines

(* === OUTPUT === *)

on startTheClock() -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        display dialog "One moment caller…" with title dialogTitle with icon
1 buttons {"OK"} default button "OK" giving up after 1
    end tell
    set startTime to current date
end startTheClock

on countdownTimer(thisStep, totalSteps, whichCuesString) -- [Shared
subroutine]
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeMSS(timeTaken)
    tell application id "com.figure53.QLab.4"
        if frontmost then
            display dialog "Time elapsed: " & timeString & " — " & thisStep
```

```
& " of " & totalSteps & " " & whichCuesString & ¬
                " done…" with title dialogTitle with icon 1 buttons
{"Cancel", "OK"} default button "OK" cancel button "Cancel" giving up after
1
        end if
    end tell
end countdownTimer

on finishedDialogBespoke()
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeNiceT(timeTaken)
    tell application "TextEdit"
        activate
        display dialog "Done." & return & return & "(That took " &
timeString & ".)" with title dialogTitle with icon 1 ¬
            buttons {"OK"} default button "OK" giving up after 60
    end tell
end finishedDialogBespoke

(* === TIME === *)

on makeMSS(howLong) -- [Shared subroutine]
    set howManyMinutes to howLong div 60
    set howManySeconds to howLong mod 60 div 1
    return (howManyMinutes as text) & ":" & my padNumber(howManySeconds, 2)
end makeMSS

on makeNiceT(howLong) -- [Shared subroutine]
    if howLong < 1 then
        return "less than a second"
    end if
    set howManyHours to howLong div 3600
    if howManyHours is 0 then
        set hourString to ""
    else if howManyHours is 1 then
        set hourString to "1 hour"
    else
        set hourString to (howManyHours as text) & " hours"
    end if
    set howManyMinutes to howLong mod 3600 div 60
    if howManyMinutes is 0 then
        set minuteString to ""
    else if howManyMinutes is 1 then
        set minuteString to "1 minute"
    else
        set minuteString to (howManyMinutes as text) & " minutes"
    end if
    set howManySeconds to howLong mod 60 div 1
    if howManySeconds is 0 then
        set secondString to ""
```

```
    else if howManySeconds is 1 then
        set secondString to "1 second"
    else
        set secondString to (howManySeconds as text) & " seconds"
    end if
    set theAmpersand to ""
    if hourString is not "" then
        if minuteString is not "" and secondString is not "" then
            set theAmpersand to ", "
        else if minuteString is not "" or secondString is not "" then
            set theAmpersand to " and "
        end if
    end if
    set theOtherAmpersand to ""
    if minuteString is not "" and secondString is not "" then
        set theOtherAmpersand to " and "
    end if
    return hourString & theAmpersand & minuteString & theOtherAmpersand &
secondString
end makeNiceT

(* === TEXT WRANGLING === *)

on padNumber(theNumber, minimumDigits) -- [Shared subroutine]
    set paddedNumber to theNumber as text
    repeat while (count paddedNumber) < minimumDigits
        set paddedNumber to "0" & paddedNumber
    end repeat
    return paddedNumber
end padNumber

on sortTextIgnoringCase(theText) -- [Shared subroutine]
    return do shell script "echo " & quoted form of theText & " | sort -f "
end sortTextIgnoringCase

(* === FILES === *)

on grabTimeForFilename() -- [Shared subroutine]
    return do shell script "date '+%y-%m-%d %H%M%S'"
end grabTimeForFilename

on makeFileFromText(newFilePath, fileContents) -- [Shared subroutine]
    copy (open for access newFilePath with write permission) to theOpenFile
    set eof theOpenFile to 0 -- Clear it out first (just in case it already
existed)
    write fileContents to theOpenFile
    close access theOpenFile
end makeFileFromText
```

Helper files

Scripts housed externally as they are best run outside of QLab.

Making

Make Group Cues from a text file [droplet]

```
(* Make Group Cues from a text file: see Description tab or theExplanation
variable

www.allthatyouhear.com | Please attribute this work if you share it, and
please report any bugs or issues you encounter

v1.0:   06/02/18        Rich Walsh (with thanks to Gareth Fry for the
original idea, years and years ago)
        16/02/18        Tested with QLab 4.4.1: no changes (although note
that "front workspace" may not be what you expect if you've hidden QLab)

<<< Last tested with QLab 4.4.1, MacOS 10.12.6 >>> *)

-- User-defined variables

property userBypassExplanation : false -- Set this to true if you don't want
to see the explanation dialog each time you run the application;
(* it's never displayed when dropping files *)

-- Explanations

property theExplanation : "This application will make Group Cues in QLab's
front workspace, based on the file you open/drop. If it has 1 column, " & ¬
    "that will be used for the names; if it has 2 they will be Cue Number
and name; 3 columns will also set Notes. Tab-delimited layouts are:

    q name
    q number     q name
    q number     q name     notes"

-- Declarations

property dialogTitle : "Make Group Cues from a text file"

global currentTIDs

on run

    set currentTIDs to AppleScript's text item delimiters
```

```
    if userBypassExplanation is false then
        display dialog theExplanation with title dialogTitle with icon 1
buttons {"Cancel", "OK"} default button "OK" cancel button "Cancel"
    end if

    -- Check QLab is running

    tell application "System Events"
        set qLabIsRunning to exists (processes whose bundle identifier is
"com.figure53.QLab.4")
    end tell
    if qLabIsRunning is false then
        my exitStrategy("QLab is not running.", 5)
        return
    end if

    -- Test for a workspace

    tell application id "com.figure53.QLab.4"
        try
            set workspaceName to q number of front workspace
        on error
            set workspaceName to false
        end try
    end tell
    if workspaceName is false then
        my exitStrategy("There is no workspace open in QLab.", 5)
        return
    end if

    set openedFile to choose file of type "public.plain-text" with prompt
"Please select a tab-delimited text file:" default location (path to
desktop)

    makeCues(openedFile)

end run

on open droppedFiles

    set currentTIDs to AppleScript's text item delimiters

    if class of droppedFiles is list then
        set droppedFiles to first item of droppedFiles -- Only handle the
first one if multiple files dropped
    end if

    -- Check dropped file is plain text

    tell application "System Events"
        set fileType to type identifier of droppedFiles
```

```
    end tell
    if fileType is "public.plain-text" then
        makeCues(droppedFiles)
    else
        my exitStrategy("That wasn't a text file!", false)
    end if

end open

-- Subroutines

(* === OUTPUT === *)

on exitStrategy(theMessage, givingUp) -- [Shared subroutine]
    if theMessage is not false then
        if givingUp is not false then
            display dialog theMessage with title dialogTitle with icon 0
buttons {"OK"} default button "OK" giving up after givingUp
        else
            display dialog theMessage with title dialogTitle with icon 0
buttons {"OK"} default button "OK"
        end if
    end if
    set AppleScript's text item delimiters to currentTIDs
end exitStrategy

(* === PROCESSING === *)

on makeCues(theFile)

    try
        set theText to read theFile
    on error
        my exitStrategy("I'm afraid that file tasted funny so I've had to
spit it out. Please check the file and try again. Sorry.", false)
        return
    end try

    set AppleScript's text item delimiters to tab

    try
        set columnCount to count (text items of paragraph 1 of theText)
    on error
        my exitStrategy("I'm afraid that file tasted funny so I've had to
spit it out. Please check the file and try again. Sorry.", false)
        return
    end try

    if columnCount > 3 then
        my exitStrategy("Too many columns…", false)
        return
```

```
    end if

    tell application id "com.figure53.QLab.4" to tell front workspace

        repeat with eachPara in paragraphs of theText

            make type "Group"

            set newCue to last item of (selected as list)

            if columnCount is 1 then

                set q name of newCue to text item 1 of eachPara

            else if columnCount is 2 then

                set q number of newCue to text item 1 of eachPara
                set q name of newCue to text item 2 of eachPara

            else if columnCount is 3 then

                set q number of newCue to text item 1 of eachPara
                set q name of newCue to text item 2 of eachPara
                set notes of newCue to text item 3 of eachPara

            end if

        end repeat

        activate

    end tell

end makeCues

(* END: Make Group Cues from a text file *)
```

Reporting

Make a text file from cues [script]

```
(* Make a text file from cues: ingest/make a tab-delimited text file and
populate (a copy of) it with a report of the cues in the current workspace
in QLab;
    see "Explanations" for further explanation. Don't forget you can do this
for the properties that have columns in the cue list view – PLUS notes &
    hotkey triggers – simply by copying a selection of cues and pasting into
TextEdit or Excel (this is also – currently – the _only_ way of getting a
list
```

```
    of hotkey triggers).

www.allthatyouhear.com | Please attribute this work if you share it, and
please report any bugs or issues you encounter

This script is not designed to be run from within QLab!

NB: if you don't have Microsoft Excel you'll need to edit the script a bit:
search for "### NO EXCEL ###"

v0.9:   12/10/09        Rich Walsh (with thanks to Jeremy Lee for some of
the basic concepts for the sister script "Make cues from a text file")
v0.9.1: 16/10/09        Now "tested" in Snow Leopard; expanded makeNiceT for
hours; fixed nasty mess with missing cue/file target strings;
        added Excel cleanup if renaming levels columns; made progress
updates more frequent; general tidying – including first attempts at
improving efficiency;
        updates for QLab 2.2.5
v0.9.2: 27/10/09        Added MSC translation; numerous typos
v1.0:   11/01/10        Fixed text-cleaning routines; "start value" is read-
only, so fixed that; byte combo (and related) origin offset; corrected minor
typos;
        added tell front workspace for elegance; wrapped text for better
wiki experience; implemented dialogTitle for cross-script pillaging
v1.1:   31/01/10        Fixed a bug (!) with carriage returns in notes
v2.0ß:  12/11/17        Major overhaul & rewrite for QLab 4… New pick'n'mix
option!
v2.0:   06/02/18        Made it possible to set userFileTimestampFormat to
false for option of no timestamps in filenames
v2.1:   07/02/18        Trapped a small bug for Excel; fixed a big bug with
not removing tabs from text properties!

Note:   "Make cues from a text file" has not been updated for QLab 4 yet…

<<< Last tested with QLab 4.1.6, MacOS 10.12.6, Microsoft Excel 16.10; NB:
**** NOT THOROUGHLY TESTED! **** >>>

-- ###FIXME### As of 4.1.6, "start time offset" for Timecode Cues can only
be reported as an unformatted number of seconds as attempting to get the
    smpte format for the non-integer rates throws an AppleScript error (they
haven't been defined as constants in the sdef yet)
-- ###FIXME### Microsoft Excel 2016 refuses to open a file as a text file
unless you open it first and then close it! It's some kind of sandbox /
permissions issue…
-- ###FIXME### Reporting solution for text format records is not very
satisfactory (suggestions welcome!)

-- ###ADD### If the "Min" level from the Audio preferences page becomes
scriptable then item 1 of userMinusInfinity could be pulled from the
workspace
```

```
*** CUSTOMISATION ***

This script has been designed to be highly customisable: you have control
over what properties are reported, what the columns are called,
how QLab's constants are displayed and some other little tweaks too.
However, you should record what you've done so you can repeat it
if the script ever gets updated, and please DON'T release your private
version into the wild!

To change what properties are reported, you can make your own text file or
add/modify "presets" below. If, say, you add one with
"set preset8 to {"file target"}" then make sure you add something to the end
of customPresets so you can choose it, and "preset8"
to the end of presetMapper so the script can find it. You'll need to add an
entry to currentListOnlyNoChildrenPresetMapper too.

To change what the columns are called, edit the relevant entry in the "set
acceptableColumnHeaders" line, eg: change "continue mode" to "f/on"
if you prefer (just don't use commas or carets – they identify level- &
gang-setting columns). You'll need to do this in any presets that refer to
that
property too, mind.

To change how QLab's constants are returned, edit the contents of the
relevant "set constantsXXX" line, eg: change "set constants11_continue_mode"
entries from "do_not_continue", "auto_continue" & "auto_follow" to "no",
"a/c" & "a/f" if you prefer.

Further tweaks are possible under "User-defined variables" & "Customisable
reporting translations"; hopefully they're self-explanatory.

*** OUTPUT FILE FORMAT ***

The final output is to a text file with the system encoding, regardless of
what type of file is ingested. Experiments suggested that this was
the safest route as problems were encountered with characters like "¶":
writing out as Unicode text leads to wrong characters in TextEdit,
whilst «class utf8» leads to wrong characters in Excel; Excel can still be
tricky with these characters if opening the file via the Finder as
it's not detecting the file origin correctly, and needs to be told that the
file is from a Macintosh. *)

-- User-defined variables

set userEscapeHatchInterval to 20 -- Set the number of cues to process
between each progress report / opportunity to cancel

set userFileTimestampFormat to "+%y-%m-%d %H%M%S" -- Set the format for
timestamps appended to filenames (set to false for none);
(* the structure of this string follows the conversion specifiers of
strftime: run "man strftime" in Terminal to see more *)
```

```
set userFileNameAppendix to "Cues report" -- String to append to files made by copying and populating existing files

set userSortPickNMix to true -- Set this to false if you don't want the list of properties in pick'n'mix mode (mostly) sorted alphabetically –
(* they'll be in the same order as acceptableColumnHeaders, which is based on QLab's AppleScript dictionary *)

set userDefaultInputs to 2 -- The default number of audio input channels to offer to report in level-reporting dialogs
set userDefaultOutputs to 32 -- The default number of outputs to offer to report in level-reporting dialogs
set userOtherRows to 1 -- The default option to offer for rows if "Other" is chosen in level-reporting dialogs
set userOtherColumns to 17 -- The default option to offer for columns if "Other" is chosen in level-reporting dialogs

set userReportCueListTimesToDisplayedPrecision to true -- Set this to false if you want times that appear in the cue list columns to be reported to
(* the 3 decimal places of the Inspector, rather than the 2 decimal places shown in the columns *)

set userTabCharacterSubstitute to "     " -- Symbol to replace tab characters stripped from text properties, as they can't be copied into tab-delimited output files
(* (not every text field will accept tab, but some surprising ones do – eg: q number) *)

set userMinusInfinity to {-120, "-INF"} -- If a level is returned below the first item, the second item in this list is substituted;
(* NB: Excel gets confused by "-INF" in the cells unless you let the script force Excel to open every column as "text"… *)

set userCarriageReturnsInLongText to "¶" -- Symbol to use for carriage returns in multi-line text properties –
(* notes, text of Text Cues, command text, script source; carriage returns have to be removed as they would corrupt the tab-delimited structure *)

set userSliceRecordColumnDelimiter to " | " -- String to use between slice time and play count in slice records
set userSliceRecordRowDelimiter to " ¶ " -- String to use between slice records when making text block
-- Sadly, within tab-delimited structure of report there is no simple way to break slice records out into multiple rows & columns

set userTextFormatDelimiter to " ; " -- String to use between coerced text formats when there is more than one
-- ###FIXME### Coercion format is not yet customisable; it's a bit like css at the moment {style:ugly; happy:not}
```

```
set userMissingCueTarget to "**** No target cue ****" -- String to return
when a cue's cue target is missing
set userMissingFileTarget to "**** No target file ****" -- String to return
when a cue's file target is missing
set userMissingQuartzFile to "None" -- String to return when a cue doesn't
have a custom Quartz Composer file (no way to detect if this matters)
set userNoMTCSyncSource to "None" -- String to return when a cue's mtc sync
source name is missing (ie: hasn't been changed from "None")

set userLayerThousandIsTop to "top" -- String to return if a cue's layer is
1000 (and hence displayed as "top" by QLab)
set userLayerZeroIsBottom to "bottom" -- String to return if a cue's layer
is 0 (and hence displayed as "bottom" by QLab)
set userConvertOpacityToPercent to true -- Report opacity as percentage (as
per Inspector) rather than 0-1 as per AppleScript property?

set userCueListsAreOrphans to "—" -- String to return when asked for the
parent of a cue list or cue cart
set userParentIsCueListBrackets to {"[", "]"} -- Symbols to surround cue's
parents if they are cue lists or carts; change to "" for none

set userIrrelevantCrosspoints to "N/A" -- String to return when asked to
examine crosspoints in rows beyond the number of audio input channels in a
cue

-- Explanations

set theExplanation1 to "This script will ingest a tab-delimited plain text
file, copy it, and then populate the copy with a report of the cues " & ¬
    "in the front workspace in QLab — returning the properties you specify
in the text file.

Alternatively, it also comes with a set of exciting presets, to which you
can add levels & gangs reporting to suit your needs each time you run the
script.

When it's done you can choose to open the file in Excel, forcing Excel to
not try to be clever and, therefore, unhelpfully disrupt the formatting. " &
¬
    "The file you get out at the end is a plain text file with no particular
application association, regardless of what you put in " & ¬
    "(this just turned out to be the easiest way).

If you choose to make your own text file then the first row must contain the
\"column headers\", ie: it should define which properties " & ¬
    "you are hoping to get from your cues. Unless you customise the script,
these column headers must match exactly the strings used in " & ¬
    "QLab's AppleScript dictionary. For a list you can look at the
dictionary, see the \"set acceptableColumnHeaders\" line inside this script,
" & ¬
    "or push the button below to copy it to the Clipboard (as a tab-
```

```
delimited row). Note that this list is slightly different from the list in
the sister script " & ¬
    "\"Make cues from a text file\", as it includes properties that are
read-only (like \"broken\") – and \"put in group\" becomes \"parent\". " & ¬
    "It's also worth noting that you probably won't be able to take the
output of this script, run it through that script and end up with the
workspace " & ¬
    "again (QLab has its own \"Save\" routine for that!).

The next page has some details about levels and customisation…"

set theExplanation2 to "Since there are currently 1,625 possible scriptable
levels for those cues that take them, it's up to you which levels to request
" & ¬
    "in your file. For any crosspoint you wish to get, add a column header
of the form \"row,column\" and those levels will be returned in that column.
" & ¬
    "For example: column header \"0,0\" specifies row 0 column 0 (ie: the
Master level), \"2,42\" would be row 2 column 42 (the crosspoint between " &
¬
    "channel 2 of your audio file and output 42). You'll get the choice to
turn some of these abstract numbers into text when the script runs " & ¬
    "(try it and find out!). It's the same syntax for gangs, but with \"^\"
instead of \",\". Be prepared to wait if you set a high number of
crosspoints to report on!

Some other substitution takes place when values are returned, eg: carriage
returns in a cue's \"notes\" are replaced with \"¶\", so as not to break " &
¬
    "the tab-delimiting. Most of the substitutions are customisable, though.

In fact, this script is highly customisable – just look inside it to find
out more.

With a few exceptions, properties that are stored as \"real\" will be
reported to the precision returned, not to the precision displayed in the
Inspector."

-- Declarations

global dialogTitle, qLabMaxAudioInputs, qLabMaxAudioChannels
set dialogTitle to "Make a text file from cues"

set qLabMaxAudioInputs to 24
set qLabMaxAudioChannels to 64

global userEscapeHatchInterval, userFileTimestampFormat,
userTabCharacterSubstitute, userCarriageReturnsInLongText, startTime

global currentTIDs
set currentTIDs to AppleScript's text item delimiters
```

```
set AppleScript's text item delimiters to "" -- Should not assume this

set acceptableColumnHeaders to {"q type", "q number", "q name", "q color",
"notes", ¬
    "cue target", "file target", "pre wait", "duration", "post wait",
"continue mode", "flagged", "autoload", "armed", "hotkey trigger", ¬
    "midi trigger", "midi command", "midi byte one", "midi byte two", "midi
byte one string", "midi byte two string", ¬
    "timecode trigger", "timecode show as timecode", "timecode hours",
"timecode minutes", "timecode seconds", "timecode frames", "timecode bits",
¬
    "wall clock trigger", "wall clock hours", "wall clock minutes", "wall
clock seconds", "mode", ¬
    "sync to timecode", "sync mode", "smpte format", "mtc sync source name",
"ltc sync channel", ¬
    "patch", "start time", "end time", "play count", "infinite loop",
"rate", "integrated fade", "lock fade to cue", ¬
    "pitch shift", "slice markers", "last slice play count", "last slice
infinite loop", ¬
    "layer", "full surface", "preserve aspect ratio", "opacity",
"translation x", "translation y", "scale x", "scale y", "do video effect",
"custom quartz file", ¬
    "hold at end", "text", "text format", "text alignment", "camera patch",
"audio fade mode", "video fade mode", "stop target when done", ¬
    "rotation type", "rotation", "do opacity", "do translation", "do
rotation", "do scale", ¬
    "command text", "always collate", "message type", "command", "channel",
"byte one", "byte two", "byte combo", "end value", ¬
    "fade", "deviceID", "command format", "command number", "q_number",
"q_list", "q_path", "macro", "control number", "control value", ¬
    "hours", "minutes", "seconds", "frames", "subframes", "send time with
set", "sysex message", "osc message type", ¬
    "q_num", "q_command", "q_params", "custom message", "udp message",
"start time offset", ¬
    "fire next cue when slice ends", "stop target when slice ends", "load
time", "assigned number", ¬
    "script source"} -- All possible cue properties (except status-related
properties, such as "running"); order is based on entries in QLab's
AppleScript dictionary

set customColumnHeaders to {"parent"} -- Additional columns this script will
understand
repeat with eachItem in customColumnHeaders
    set end of acceptableColumnHeaders to eachItem as text
end repeat

set reportingOnlyColumns to {"unique ID", "q list name", "q display name",
"q default name", "broken", "audio input channels", "start value"}
-- These are r/o properties that can't be set/made
repeat with eachItem in reportingOnlyColumns
    set end of acceptableColumnHeaders to eachItem as text
```

```
end repeat

set levelColumns to {} -- If a column header contains "," it will be added
to this list
set gangColumns to {} -- If a column header contains "^" it will be added to
this list

-- This is a list of properties that can be read for each cue type (do not
edit these variables); ^ indicates text properties that testing has shown
can include tabs

-- set index1_q_type: every cue has a q type (text)
-- set index2_q_number: every cue has q number (text^)
-- set index3_q_name: every cue has q name (text^)
-- set index4_q_color: every cue has q color (text)
-- set index5_notes: every cue has notes (text^)
set index6_cue_target to {"Fade", "Start", "Stop", "Pause", "Load", "Reset",
"Devamp", "GoTo", "Target", "Arm", "Disarm"} -- (cue / "missing value");
(* although every cue returns a value for cue target, these are the only
types that actually take one *)
set index7_file_target to {"Audio", "Video", "MIDI File"} -- (file /
"missing value");
(* although every cue returns a value for file target, these are the only
types that actually take one *)
-- set index8_pre_wait: every cue has pre wait (real)
set index9_duration to {"Group", "Audio", "Video", "Text", "Light", "Fade",
"Network", "MIDI", "MIDI File", "Wait"} -- (real);
(* although every cue returns a value for duration, for the other types it
makes no sense to record "0" *)
-- set index10_post_wait: every cue has post wait (real)
-- set index11_continue_mode: every cue has continue mode (constants)
-- set index12_flagged: every cue has flagged (boolean)
-- set index13_autoload: every cue has autoload (boolean)
-- set index14_armed: every cue has armed (boolean)
-- set index15_hotkey_trigger: every cue has hotkey trigger (constants)
-- set index16_midi_trigger: every cue has midi trigger (constants)
-- set index17_midi_command: every cue has midi command (constants)
-- set index18_midi_byte_one: every cue has midi byte one (integer)
-- set index19_midi_byte_two: every cue has midi byte two (integer)
-- set index20_midi_byte_one_string: every cue has midi byte one string
(text)
-- set index21_midi_byte_two_string: every cue has midi byte two string
(text)
-- set index22_timecode_trigger: every cue has timecode trigger (constants)
-- set index23_timecode_show_as_timecode: every cue has timecode show as
timecode (boolean)
-- set index24_timecode_hours: every cue has timecode hours (integer)
-- set index25_timecode_minutes: every cue has timecode minutes (integer)
-- set index26_timecode_seconds: every cue has timecode seconds (integer)
-- set index27_timecode_frames: every cue has timecode frames (integer)
-- set index28_timecode_bits: every cue has timecode bits (integer)
```

```
-- set index29_wall_clock_trigger: every cue has wall clock trigger
(constants)
-- set index30_wall_clock_hours: every cue has wall clock hours (integer);
NB: value is reported as 24-hour regardless of display choice in Inspector
-- set index31_wall_clock_minutes: every cue has wall clock minutes
(integer)
-- set index32_wall_clock_seconds: every cue has wall clock seconds
(integer)
set index33_mode to {"Cue List", "Group"} -- (constants)
set index34_sync_to_timecode to {"Cue List"} -- (constants)
set index35_sync_mode to {"Cue List"} -- (constants)
set index36_smpte_format to {"Cue List", "MIDI"} -- (constants) ###FIXME###
"Timecode" should be in index36_smpte_format,
(* but as of 4.1.6, QLab throws error -10000 if you try to get smpte format
from a Timecode Cue… *)
set index37_mtc_sync_source_name to {"Cue List"} -- (text^)
set index38_ltc_sync_channel to {"Cue List"} -- (integer)
set index39_patch to {"Audio", "Mic", "Video", "Network", "MIDI", "MIDI
File", "Timecode"} -- (integer) ###FIXME### As of 4.1.6, "Text" has entry in
sdef
set index40_start_time to {"Audio", "Video"} -- (real) ###FIXME### As of
4.1.6, "Text" has entry in sdef
set index41_end_time to {"Audio", "Video"} -- (real) ###FIXME### As of
4.1.6, "Text" has entry in sdef
set index42_play_count to {"Audio", "Video"} -- (integer) ###FIXME### As of
4.1.6, "Text" has entry in sdef
set index43_infinite_loop to {"Audio", "Video"} -- (boolean) ###FIXME### As
of 4.1.6, "Text" has entry in sdef
set index44_rate to {"Audio", "Video", "MIDI File"} -- (real) ###FIXME### As
of 4.1.6, "Text" has entry in sdef
set index45_integrated_fade to {"Audio", "Video"} -- (constants) ###FIXME###
As of 4.1.6, "Text" has entry in sdef
set index46_lock_fade_to_cue to {"Audio", "Video"} -- (constants)
###FIXME### As of 4.1.6, "Text" has entry in sdef
set index47_pitch_shift to {"Audio", "Video"} -- (constants) ###FIXME### As
of 4.1.6, "Text" has entry in sdef
set index48_slice_markers to {"Audio", "Video"} -- (records) ###FIXME### As
of 4.1.6, "Text" has entry in sdef
###TODO### Having slice markers in a single column may not scale well to
"Make cues from a text file" when (if) it is updated…
set index49_last_slice_play_count to {"Audio", "Video"} -- (integer)
###FIXME### As of 4.1.6, "Text" has entry in sdef
set index50_last_slice_infinite_loop to {"Audio", "Video"} -- (boolean)
###FIXME### As of 4.1.6, "Text" has entry in sdef
set index51_layer to {"Video", "Camera", "Text"} -- (integer)
set index52_full_surface to {"Video", "Camera", "Text"} -- (boolean)
set index53_preserve_aspect_ratio to {"Video", "Camera", "Text", "Fade"} --
(boolean)
set index54_opacity to {"Video", "Camera", "Text", "Fade"} -- (real); NB:
reports as 0-1 but displays as 0-100%
set index55_translation_x to {"Video", "Camera", "Text", "Fade"} -- (real)
```

```
set index56_translation_y to {"Video", "Camera", "Text", "Fade"} -- (real)
set index57_scale_x to {"Video", "Camera", "Text", "Fade"} -- (real)
set index58_scale_y to {"Video", "Camera", "Text", "Fade"} -- (real)
set index59_do_video_effect to {"Video", "Camera", "Text"} -- (boolean)
set index60_custom_quartz_file to {"Video", "Camera", "Text"} -- (file)
set index61_hold_at_end to {"Video", "Text"} -- (boolean)
set index62_text to {"Text"} -- (text^)
set index63_text_format to {"Text"} -- (records); wordIndex (integer) can be
used as a record label when setting, but not getting
###TODO### Handling text format records will take some considerable thought
in "Make cues from a text file" when (if) it is updated!
set index64_text_alignment to {"Text"} -- (text)
set index65_camera_patch to {"Camera"} -- (integer)
set index66_audio_fade_mode to {"Fade"} -- (constants)
set index67_video_fade_mode to {"Fade"} -- (constants)
set index68_stop_target_when_done to {"Fade"} -- (boolean)
set index69_rotation_type to {"Fade"} -- (integer)
set index70_rotation to {"Fade"} -- (real)
set index71_do_opacity to {"Fade"} -- (boolean)
set index72_do_translation to {"Fade"} -- (boolean)
set index73_do_rotation to {"Fade"} -- (boolean)
set index74_do_scale to {"Fade"} -- (boolean)
set index75_command_text to {"Light"} -- (text^)
set index76_always_collate to {"Light"} -- (boolean)
set index77_message_type to {"MIDI"} -- (constants)
set index78_command to {"MIDI"} -- (constants)
set index79_channel to {"MIDI"} -- (integer)
set index80_byte_one to {"MIDI"} -- (integer)
set index81_byte_two to {"MIDI"} -- (integer)
set index82_byte_combo to {"MIDI"} -- (integer)
set index83_end_value to {"MIDI"} -- (integer)
set index84_fade to {"MIDI"} -- (constants)
set index85_deviceID to {"MIDI"} -- (integer)
set index86_command_format to {"MIDI"} -- (integer)
set index87_command_number to {"MIDI"} -- (integer)
set index88_q__number to {"MIDI"} -- (text)
set index89_q__list to {"MIDI"} -- (text)
set index90_q__path to {"MIDI"} -- (text)
set index91_macro to {"MIDI"} -- (integer)
set index92_control_number to {"MIDI"} -- (integer)
set index93_control_value to {"MIDI"} -- (integer)
set index94_hours to {"MIDI"} -- (integer)
set index95_minutes to {"MIDI"} -- (integer)
set index96_seconds to {"MIDI"} -- (integer)
set index97_frames to {"MIDI"} -- (integer)
set index98_subframes to {"MIDI"} -- (integer)
set index99_send_time_with_set to {"MIDI"} -- (boolean)
set index100_sysex_message to {"MIDI"} -- (text^)
set index101_osc_message_type to {"Network"} -- (constants)
set index102_q_num to {"Network"} -- (text^)
set index103_q_command to {"Network"} -- (number)
```

```
set index104_q_params to {"Network"} -- (text^)
set index105_custom_message to {"Network"} -- (text^)
set index106_udp_message to {"Network"} -- (text^)
set index107_start_time_offset to {"Timecode"} -- (real)
set index108_fire_next_cue_when_slice_ends to {"Devamp"} -- (boolean)
set index109_stop_target_when_slice_ends to {"Devamp"} -- (boolean)
set index110_load_time to {"Load"} -- (real)
set index111_assigned_number to {"Target"} -- (text^)
set index112_script_source to {"Script"} -- (text^)
-- set index113_parent: every cue has parent (cue)
-- set index114_unique_ID: every cue has unique ID (text)
-- set index115_q_list_name: every cue has q list name (text^)
-- set index116_q_display_name: every cue has q display name (text^)
-- set index117_q_default_name: every cue has q default name (text^)
-- set index118_broken: every cue has broken (boolean)
set index119_audio_input_channels to {"Audio", "Mic", "Video"} -- (integer)
###FIXME### As of 4.1.6, "Text" has entry in sdef
set index120_start_value to {"MIDI"} -- (integer)

set index_takesLevel to {"Audio", "Video", "Fade"} -- Special private index
for custom column headers
set index_takesGang to {"Audio", "Video", "Fade"} -- Special private index
for custom column headers

-- This is a list of values for any constants (which can be used to
customise the entries returned in the text file)

set constants11_continue_mode to {"do_not_continue", "auto_continue",
"auto_follow"}
set constants12_flagged to {"true", "false"}
set constants13_autoload to {"true", "false"}
set constants14_armed to {"true", "false"}
set constants15_hotkey_trigger to {"enabled", "disabled"}
set constants16_midi_trigger to {"enabled", "disabled"}
set constants17_midi_command to {"note_on", "note_off", "program_change",
"control_change", "key_pressure", "channel_pressure"}
set constants22_timecode_trigger to {"enabled", "disabled"}
set constants23_timecode_show_as_timecode to {"true", "false"}
set constants29_wall_clock_trigger to {"enabled", "disabled"}
set constants33_mode to {"cue_list", "fire_first_enter_group",
"fire_first_go_to_next_cue", "fire_all", "fire_random"}
set constants34_sync_to_timecode to {"enabled", "disabled"}
set constants35_sync_mode to {"mtc", "ltc"}
set constants36_smpte_format to {"fps_24", "fps_25", "fps_30_drop",
"fps_30_non_drop"}
set constants43_infinite_loop to {"true", "false"}
set constants45_integrated_fade to {"enabled", "disabled"}
set constants46_lock_fade_to_cue to {"enabled", "disabled"}
set constants47_pitch_shift to {"enabled", "disabled"}
set constants50_last_slice_infinite_loop to {"true", "false"}
set constants52_full_surface to {"true", "false"}
```

```
set constants53_preserve_aspect_ratio to {"true", "false"}
set constants59_do_video_effect to {"true", "false"}
set constants61_hold_at_end to {"true", "false"}
set constants66_audio_fade_mode to {"absolute", "relative"}
set constants67_video_fade_mode to {"absolute", "relative"}
set constants68_stop_target_when_done to {"true", "false"}
set constants71_do_opacity to {"true", "false"}
set constants72_do_translation to {"true", "false"}
set constants73_do_rotation to {"true", "false"}
set constants74_do_scale to {"true", "false"}
set constants76_always_collate to {"true", "false"}
set constants77_message_type to {"voice", "msc", "sysex"}
set constants78_command to {"note_on", "note_off", "program_change",
"control_change", "key_pressure", "channel_pressure", "pitch_bend"}
set constants84_fade to {"enabled", "disabled"}
set constants99_send_time_with_set to {"true", "false"}
set constants101_osc_message_type to {"qlab", "custom", "udp"}
set constants108_fire_next_cue_when_slice_ends to {"true", "false"}
set constants109_stop_target_when_slice_ends to {"true", "false"}
set constants118_broken to {"true", "false"}

-- This variable is used to translate rotation type integers into English

set translation69_rotation_type to {0, "3D orientation", 1, "X rotation", 2,
"Y rotation", 3, "Z rotation"}

-- These variables are used to translate MSC integers into English

set translation86_command_format to {1, "Lighting (General)", 2, "Moving
Lights", 3, "Color Changers", 4, "Strobes", 5, "Lasers", 6, "Chasers", ¬
    16, "Sound (General)", 17, "Music", 18, "CD Players", 19, "EPROM
Playback", 20, "Audio Tape Machines", 21, "Intercoms", 22, "Amplifiers", ¬
    23, "Audio Effects Devices", 24, "Equalizers", 32, "Machinery
(General)", 33, "Rigging", 34, "Flys", 35, "Lifts", 36, "Turntables", 37,
"Trusses", ¬
    38, "Robots", 39, "Animation", 40, "Floats", 41, "Breakaways", 42,
"Barges", 48, "Video (General)", 49, "Video Tape Machines", ¬
    50, "Video Cassette Machines", 51, "Video Disc Players", 52, "Video
Switchers", 53, "Video Effects", 54, "Video Character Generators", ¬
    55, "Video Still Stores", 56, "Video Monitors", 64, "Projection
(General)", 65, "Film Projectors", 66, "Slide Projectors", 67, "Video
Projectors", ¬
    68, "Dissolvers", 69, "Shutter Controls", 80, "Process Control
(General)", 81, "Hydraulic Oil", 82, "H2O", 83, "CO2", 84, "Compressed Air",
¬
    85, "Natural Gas", 86, "Fog", 87, "Smoke", 88, "Cracked Haze", 96,
"Pyrotechnics (General)", 97, "Fireworks", 98, "Explosions", 99, "Flame", ¬
    100, "Smoke Pots", 127, "All Types"}
set translation87_command_number to {1, "GO", 2, "STOP", 3, "RESUME", 4,
"TIMED_GO", 5, "LOAD", 6, "SET", 7, "FIRE", 8, "ALL_OFF", ¬
    9, "RESTORE", 10, "RESET", 11, "GO_OFF", 16, "GO/JAM_CLOCK", 17,
```

```
"STANDBY_+", 18, "STANDBY_-", 19, "SEQUENCE_+", ¬
    20, "SEQUENCE_-", 21, "START_CLOCK", 22, "STOP_CLOCK", 23, "ZERO_CLOCK",
24, "SET_CLOCK", 25, "MTC_CHASE_ON", ¬
    26, "MTC_CHASE_OFF", 27, "OPEN_CUE_LIST", 28, "CLOSE_CUE_LIST", 29,
"OPEN_CUE_PATH", 30, "CLOSE_CUE_PATH"}

-- This variable is used to translate QLab type Network message command
integers into English

set translation103_q_command to {1, "start", 2, "stop", 3, "hardStop", 4,
"pause", 5, "resume", 6, "togglePause", 7, "load", 8, "preview", 9, "reset",
¬
    10, "panic", 11, "number", 12, "name", 13, "notes", 14,
"cueTargetNumber", 15, "preWait", 16, "duration", 17, "postWait", 18,
"continueMode", ¬
    19, "flagged", 20, "armed", 21, "colorName"}

-- Customisable reporting translations

set levelCrosspointsInEnglish to {"0,0", "Master", "0,", "Output "} --
Replacement text for levels columns
repeat with i from 1 to qLabMaxAudioInputs
    set end of levelCrosspointsInEnglish to (i & ",0") as text
    set end of levelCrosspointsInEnglish to ("Input " & i) as text
end repeat

set gangCrosspointsInEnglish to {"0^0", "Gang: Master", "0^", "Gang: Output
"} -- Replacement text for gangs columns
repeat with i from 1 to qLabMaxAudioInputs
    set end of gangCrosspointsInEnglish to (i & "^0") as text
    set end of gangCrosspointsInEnglish to ("Gang: Input " & i) as text
end repeat

-- Reporting presets (customisable)

set customPresets to {"The columns that you see", "Properties for audio &
MIDI", "Properties for video", "All the properties", "Just the basic facts",
¬
    "Key properties for a levels report", "Top level cue details of current
cue list only"}

set preset1 to {"unique ID", "q type", "q number", "q list name", "cue
target", "file target", "pre wait", "duration", "post wait", "continue
mode", "broken", ¬
    "notes", "mode", "parent"} -- The columns that you see
set preset2 to {"unique ID", "q type", "q number", "q list name", "cue
target", "file target", "pre wait", "duration", "post wait", "continue
mode", "broken", ¬
    "notes", "mode", "parent", "patch", "audio input channels", "rate",
"pitch shift", "start time", "end time", "play count", "infinite loop", ¬
    "integrated fade", "lock fade to cue", "slice markers", "last slice play
```

```
count", "last slice infinite loop", ¬
    "audio fade mode", "stop target when done", "message type", "command",
"channel", "byte one", "byte two", "byte combo", "start value", "end value",
¬
    "fade", "SMPTE format", "device ID", "command format", "command number",
"q_number", "q_list", "q_path", "macro", "control number", "control value",
¬
    "hours", "minutes", "seconds", "frames", "subframes", "send time with
set", "sysex message"} -- Properties for audio & MIDI
set preset3 to {"unique ID", "q type", "q number", "q list name", "cue
target", "file target", "pre wait", "duration", "post wait", "continue
mode", "broken", ¬
    "notes", "mode", "parent", "patch", "rate", "start time", "end time",
"play count", "infinite loop", "layer", "full surface", "preserve aspect
ratio", "opacity", ¬
    "translation x", "translation y", "scale x", "scale y", "do video
effect", "custom quartz file", "hold at end", "video fade mode", "stop
target when done", ¬
    "rotation type", "rotation", "do opacity", "do translation", "do
rotation", "do scale"} -- Properties for video
set preset4 to acceptableColumnHeaders -- All the properties
set preset5 to {"q number", "q list name", "continue mode", "notes", "mode",
"parent"} -- Can you show me where it hurts
set preset6 to {"unique ID", "q type", "q number", "q list name", "cue
target", "file target", "broken", "notes", ¬
    "audio input channels", "patch"} -- Key properties for a levels report
set preset7 to {"q number", "q list name", "continue mode", "notes"} -- Top
level cue details of current cue list only

set presetMapper to {preset1, preset2, preset3, preset4, preset5, preset6,
preset7}
set currentListOnlyNoChildrenPresetMapper to {preset1:false, preset2:false,
preset3:false, preset4:false, preset5:false, preset6:false, preset7:true} --
If true,
(* the preset will only report cues directly in the current cue list, not
their children *)

-- Add fixed presets

set availablePresets to {"I've rolled my own"} & customPresets &
{"Pick'n'mix"}

-- General variables

set currentListOnlyNoChildren to false
set propertiesToColumns to {}
set propertiesToColumnsRef to a reference to propertiesToColumns
set headerRow to {}
set headerRowRef to a reference to headerRow

try -- This overall try makes sure TIDs are reset and open files are closed
```

```
if any "Cancel" button is pushed

    -- Preamble

    set theNavigator to "Review instructions"
    repeat until theNavigator is "Get on with it"
        set theNavigator to button returned of (display dialog "Would you
like to review the instructions for this script?" with title dialogTitle ¬
            with icon 1 buttons {"Review instructions", "Cancel", "Get on
with it"} default button "Get on with it" cancel button "Cancel")
        if theNavigator is "Review instructions" then
            set finishedReading to false
            repeat until finishedReading is true
                set instructionButton1 to "Copy headers to Clipboard"
                repeat until instructionButton1 is not "Copy headers to
Clipboard"
                    set instructionButton1 to button returned of (display
dialog theExplanation1 with title dialogTitle with icon 1 ¬
                        buttons {"Back to start", "Copy headers to
Clipboard", "To page 2 >>>"} default button "To page 2 >>>")
                    if instructionButton1 is "Copy headers to Clipboard"
then
                        set the clipboard to
listToDelimitedText(acceptableColumnHeaders, tab)
                    else if instructionButton1 is "Back to start" then
                        set finishedReading to true
                    end if
                end repeat
                if instructionButton1 is "To page 2 >>>" then
                    set instructionButton2 to "Copy headers to Clipboard"
                    repeat until instructionButton2 is not "Copy headers to
Clipboard"
                        set instructionButton2 to button returned of
(display dialog theExplanation2 with title dialogTitle with icon 1 ¬
                            buttons {"<<< To page 1", "Copy headers to
Clipboard", "Back to start"} default button "Back to start")
                        if instructionButton2 is "Copy headers to Clipboard"
then
                            set the clipboard to
listToDelimitedText(acceptableColumnHeaders, tab)
                        else if instructionButton2 is "Back to start" then
                            set finishedReading to true
                        end if
                    end repeat
                end if
            end repeat
        end if

    end repeat

    -- Check QLab is running
```

```
    tell application "System Events"
        set qLabIsRunning to exists (processes whose bundle identifier is
"com.figure53.QLab.4")
    end tell
    if qLabIsRunning is false then
        exitStrategy("QLab is not running.", 5)
        return
    end if

    -- Test for a workspace (get the name at the same time)

    tell application id "com.figure53.QLab.4"
        try
            set workspaceName to q number of front workspace
        on error
            set workspaceName to false
        end try
    end tell
    if workspaceName is false then
        exitStrategy("There is no workspace open in QLab.", 5)
        return
    end if

    -- Select a preset

    choose from list availablePresets with prompt "Please choose whether to
ingest your own file, or use one of my preset reports:" with title
dialogTitle ¬
        default items item 1 of availablePresets
    if result is not false then
        set presetMenuChoice to item 1 of result
    else
        exitStrategy(false, false)
        return
    end if

    repeat with i from 2 to (count availablePresets) - 1
        if presetMenuChoice is item i of availablePresets then
            set chosenPreset to item (i - 1) of presetMapper
            set currentListOnlyNoChildren to item (i - 1) of
(currentListOnlyNoChildrenPresetMapper as list)
            exit repeat
        end if
    end repeat

    -- Deal with pick'n'mix choice…

    if presetMenuChoice is last item of availablePresets then
        if userSortPickNMix is true then
            set pinnedToTop to items 1 thru 3 of acceptableColumnHeaders
```

```
            set sortableHeaders to items 4 thru end of
acceptableColumnHeaders
            set sortedHeaders to paragraphs of
sortTextIgnoringCase(listToDelimitedText(sortableHeaders, linefeed))
            set pickNChoose to pinnedToTop & sortedHeaders
        else
            set pickNChoose to acceptableColumnHeaders
        end if
        choose from list pickNChoose with prompt "Please choose the
properties to report on:" with title dialogTitle with multiple selections
allowed
        if result is not false then
            set chosenPreset to result
            offerToSave("Would you like to save that selection to the
Desktop as a text file?", ¬
                {"Yes", "No", "Cancel"}, "No", "Cancel", "Yes",
listToDelimitedText(chosenPreset, tab), "Pick'n'mix selection")
        else
            exitStrategy(false, false)
            return
        end if
    end if

    -- Prepare the output file…

    if userFileTimestampFormat is false then
        set fileTimeString to ""
    else
        set fileTimeString to " | " & grabTimeForFilenameBespoke()
    end if

    -- …either get the file (and prepare to copy it)…

    if presetMenuChoice is first item of availablePresets then

        set originalFile to choose file of type "public.plain-text" with
prompt "Please select a tab-delimited text file which contains a header row
" & ¬
            "for the properties you wish to include in the report:" default
location (path to desktop)

        try
            set theText to read originalFile
        on error
            exitStrategy("I'm afraid that file tasted funny so I've had to
spit it out. Please check the file and try again. Sorry.", false)
            return
        end try

        tell application "System Events"
            set theContainer to path of container of originalFile as text
```

```
            set theExtension to name extension of originalFile
            if theExtension is "" then
                set theName to name of originalFile
            else
                set theFullName to name of originalFile
                set theName to text 1 through (-1 - ((length of
theExtension) + 1)) of theFullName
                set theExtension to "." & theExtension
            end if
            set outputFilename to theName & " | QLab | " & workspaceName & "
| " & userFileNameAppendix & fileTimeString & theExtension
            set outputFile to theContainer & outputFilename
        end tell

        checkForFile(outputFile)

        set AppleScript's text item delimiters to tab
        try
            set headerRowRef to text items of paragraph 1 of theText -- Pull
headers from file
        on error
            exitStrategy("I'm afraid that file tasted funny so I've had to
spit it out. Please check the file and try again. Sorry.", false)
            return
        end try
        set AppleScript's text item delimiters to ""

        -- …or prepare to make it on the Desktop, with current time appended
to name

    else

        set outputFilename to "QLab | " & workspaceName & " | " &
dialogTitle & fileTimeString & ".txt"
        set outputFile to ((path to desktop) as text) & outputFilename

        checkForFile(outputFile)

        set headerRowRef to chosenPreset

        -- Ask about levels

        if currentListOnlyNoChildren is false then -- Don't ask about levels
for currentListOnlyNoChildren preset

            if class of userDefaultInputs is not integer or
userDefaultInputs < 0 or userDefaultInputs > qLabMaxAudioInputs or ¬
                class of userDefaultOutputs is not integer or
userDefaultOutputs < 0 or userDefaultOutputs > qLabMaxAudioChannels then
                exitStrategy("There is a problem with the user-defined
level-reporting dialog variables.", false)
```

```
                return
            end if

            if userDefaultInputs is 1 then
                set defaultOffer to "Mono to "
            else if userDefaultInputs is 2 then
                set defaultOffer to "Stereo to "
            else
                set defaultOffer to userDefaultInputs & " inputs to "
            end if
            if userDefaultOutputs is 1 then
                set defaultOffer to defaultOffer & "1 output"
            else
                set defaultOffer to defaultOffer & userDefaultOutputs & "
outputs"
            end if

            set levelsQuestion to button returned of (display dialog "Do you
want to include any levels?" with title dialogTitle with icon 1 ¬
                buttons {"No", defaultOffer, "Other"} default button
defaultOffer)
            if levelsQuestion is defaultOffer then
                set {totalRows, totalColumns} to {userDefaultInputs,
userDefaultOutputs}
            else if levelsQuestion is "Other" then
                set totalRows to
(enterANumberWithRangeWithCustomButton("Enter the number of rows you wish to
report " & ¬
                    "(ie: a number between 1 & " & qLabMaxAudioInputs + 1 &
"):", userOtherRows, 1, true, ¬
                    qLabMaxAudioInputs + 1, true, true, {}, "OK")) - 1
                set totalColumns to
(enterANumberWithRangeWithCustomButton("Enter the number of columns you wish
to report " & ¬
                    "(ie: a number between 1 & " & qLabMaxAudioChannels + 1
& "):", userOtherColumns, 1, true, ¬
                    qLabMaxAudioChannels + 1, true, true, {}, "OK")) - 1
            end if
            if levelsQuestion is not "No" then
                repeat with i from 0 to totalRows
                    repeat with j from 0 to totalColumns
                        set end of headerRowRef to (i & "," & j) as text
                    end repeat
                end repeat
                if button returned of (display dialog "Do you want to report
gangs for those crosspoints too?" with title dialogTitle with icon 1 ¬
                    buttons {"No", "Yes"} default button "Yes") is "Yes"
then
                    repeat with i from 0 to totalRows
                        repeat with j from 0 to totalColumns
                            set end of headerRowRef to (i & "^" & j) as text
```

```
                        end repeat
                    end repeat
                    set saveMessagePrefix to "Levels & gangs"
                    set saveAppendix to "Custom preset with levels & gangs"
                else
                    set saveMessagePrefix to "Levels"
                    set saveAppendix to "Custom preset with levels"
                end if
                offerToSave(saveMessagePrefix & ¬
                    " reporting columns added. Would you like to save the
augmented preset to the Desktop as a text file for re-use?", ¬
                    {"Yes", "No", "Cancel"}, "No", "Cancel", "Yes",
listToDelimitedText(headerRowRef, tab), saveAppendix)
            end if

        end if

        set theText to listToDelimitedText(headerRowRef, tab)

    end if

    -- Set up translation matrices

    repeat with i from 1 to count acceptableColumnHeaders -- Find which
properties are in text file, and which column they are in
        set end of propertiesToColumnsRef to 0
        repeat with j from 1 to count headerRowRef
            if item j of headerRowRef is item i of acceptableColumnHeaders
then
                set item i of propertiesToColumnsRef to j
            end if
        end repeat
    end repeat

    repeat with i from 1 to count headerRowRef -- Make lists of all columns
flagged as levels/gangs
        if item i of headerRowRef contains "," then
            set end of levelColumns to i
        end if
        if item i of headerRowRef contains "^" then
            set end of gangColumns to i
        end if
    end repeat

    -- Make levels/gangs columns headers easier to read

    set AppleScript's text item delimiters to tab

    if (count levelColumns) is not 0 then
        if button returned of (display dialog ¬
            "Now that I've got what I need from the headers for levels &
```

```
gangs reporting, shall I translate (some of) them into text for you?" with
title ¬
            dialogTitle with icon 1 buttons {"No thanks", "Please do"}
default button "Please do") is "Please do" then
            set dirtyHeader to paragraph 1 of theText -- Only work on first
row (in case there is more text in the file)
            set theRest to rest of paragraphs of theText
            set dirtyColumns to text items of dirtyHeader
            set cleanColumns to {}
            repeat with eachColumn in dirtyColumns
                if eachColumn contains "," then -- Only work on levels
columns (gangs columns aren't escaped as there are no caret-separated
files!)
                    set AppleScript's text item delimiters to "\"" -- Strip
out Excel formatting: 0,0 becomes "0,0" when exported from Excel
                    set cleanStore to text items of eachColumn
                    set AppleScript's text item delimiters to ""
                    set end of cleanColumns to cleanStore as text
                else
                    set end of cleanColumns to eachColumn as text
                end if
            end repeat
            set AppleScript's text item delimiters to tab
            set cleanHeader to cleanColumns as text
            repeat with i from 1 to count levelCrosspointsInEnglish by 2 --
Replace strings in header row
                set AppleScript's text item delimiters to item i of
levelCrosspointsInEnglish
                set englishHeader to text items of cleanHeader
                set AppleScript's text item delimiters to item (i + 1) of
levelCrosspointsInEnglish
                set cleanHeader to englishHeader as text
            end repeat
            repeat with i from 1 to count gangCrosspointsInEnglish by 2 --
Replace strings in header row
                set AppleScript's text item delimiters to item i of
gangCrosspointsInEnglish
                set englishHeader to text items of cleanHeader
                set AppleScript's text item delimiters to item (i + 1) of
gangCrosspointsInEnglish
                set cleanHeader to englishHeader as text
            end repeat
            if (count theRest) is 0 then
                set theText to cleanHeader
            else
                set AppleScript's text item delimiters to return
                set theText to cleanHeader & return & theRest as text --
Stitch the file back together
            end if
        end if
    end if
```

```
    set AppleScript's text item delimiters to ""

    -- Now, to business

    startTheClock()

    tell application id "com.figure53.QLab.4"

        activate

        tell front workspace

            -- Get the cues, subject to currentListOnlyNoChildren

            if currentListOnlyNoChildren is false then
                set allCues to cues
            else
                set allCues to cues of current cue list
            end if
            set allCuesRef to a reference to allCues
            set countCues to count allCuesRef

            -- Step through cues

            repeat with i from 1 to countCues

                set eachCue to item i of allCuesRef

                -- Prepare a list to hold the properties for this cue

                set theProperties to {}

                repeat (count headerRowRef) times
                    set end of theProperties to ""
                end repeat

                set thePropertiesRef to (a reference to theProperties)

                -- Get the type

                set theType to q type of eachCue as text
                if theType is "Group" and mode of eachCue is cue_list then -
- Cue carts have their own q type, but cue lists don't!
                    set theType to "Cue List"
                end if

                -- These if…then clauses retrieve the relevant property on
each cue based on its type and the contents of the text file

                if item 1 of propertiesToColumnsRef is not 0 then --
```

```
index1_q_type
                    set item (item 1 of propertiesToColumnsRef) of
thePropertiesRef to theType
                end if

                if item 2 of propertiesToColumnsRef is not 0 then --
index2_q_number
                    set theItem to q number of eachCue as text
                    set item (item 2 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 3 of propertiesToColumnsRef is not 0 then --
index3_q_name
                    set theItem to q name of eachCue as text
                    set item (item 3 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 4 of propertiesToColumnsRef is not 0 then --
index4_q_color
                    set theItem to q color of eachCue as text
                    set item (item 4 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 5 of propertiesToColumnsRef is not 0 then --
index5_notes
                    set theItem to notes of eachCue as text
                    set item (item 5 of propertiesToColumnsRef) of
thePropertiesRef to my noReturns(my noTabs(theItem))
                end if

                if item 6 of propertiesToColumnsRef is not 0 and theType is
in index6_cue_target then
                    set theItem to cue target of eachCue
                    if theItem is missing value then
                        set targetTitle to userMissingCueTarget
                    else
                        set targetTitle to (q list name of theItem) as text
                        if targetTitle is "" then
                            set targetTitle to (q number of theItem) as text
                            if targetTitle is "" then
                                set targetTitle to "id: " & (uniqueID of
theItem) as text
                            end if
                        end if
                    end if
                    set item (item 6 of propertiesToColumnsRef) of
thePropertiesRef to targetTitle
                end if
```

```
                if item 7 of propertiesToColumnsRef is not 0 and theType is
in index7_file_target then
                    set theItem to file target of eachCue
                    if theItem is missing value then
                        set targetTitle to userMissingFileTarget
                    else
                        set targetTitle to POSIX path of theItem as text
                    end if
                    set item (item 7 of propertiesToColumnsRef) of
thePropertiesRef to targetTitle
                end if

                if item 8 of propertiesToColumnsRef is not 0 then --
index8_pre_wait
                    set theItem to pre wait of eachCue
                    if userReportCueListTimesToDisplayedPrecision is true
then
                        set item (item 8 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSss(theItem)
                    else
                        set item (item 8 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSsss(theItem)
                    end if
                end if

                if item 9 of propertiesToColumnsRef is not 0 and theType is
in index9_duration then
                    set theItem to duration of eachCue
                    if theType is not "Group" or mode of eachCue is fire_all
then -- QLab only calculates duration for fire_all groups
                        if userReportCueListTimesToDisplayedPrecision is
true then
                            set item (item 9 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSss(theItem)
                        else
                            set item (item 9 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSsss(theItem)
                        end if
                    end if
                end if

                if item 10 of propertiesToColumnsRef is not 0 then --
index10_post_wait
                    set theItem to post wait of eachCue
                    if userReportCueListTimesToDisplayedPrecision is true
then
                        set item (item 10 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSss(theItem)
                    else
                        set item (item 10 of propertiesToColumnsRef) of
```

```
thePropertiesRef to my makeHHMMSSsss(theItem)
                    end if
                end if

                if item 11 of propertiesToColumnsRef is not 0 then --
index11_continue_mode
                    set theItem to continue mode of eachCue
                    if theItem is do_not_continue then
                        set item (item 11 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants11_continue_mode
                    else if theItem is auto_continue then
                        set item (item 11 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants11_continue_mode
                    else if theItem is auto_follow then
                        set item (item 11 of propertiesToColumnsRef) of
thePropertiesRef to item 3 of constants11_continue_mode
                    end if
                end if

                if item 12 of propertiesToColumnsRef is not 0 then --
index12_flagged
                    set theItem to flagged of eachCue
                    if theItem is true then
                        set item (item 12 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants12_flagged
                    else if theItem is false then
                        set item (item 12 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants12_flagged
                    end if
                end if

                if item 13 of propertiesToColumnsRef is not 0 then --
index13_autoload
                    set theItem to autoload of eachCue
                    if theItem is true then
                        set item (item 13 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants13_autoload
                    else if theItem is false then
                        set item (item 13 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants13_autoload
                    end if
                end if

                if item 14 of propertiesToColumnsRef is not 0 then --
index14_armed
                    set theItem to armed of eachCue
                    if theItem is true then
                        set item (item 14 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants14_armed
                    else if theItem is false then
                        set item (item 14 of propertiesToColumnsRef) of
```

```
thePropertiesRef to item 2 of constants14_armed
                    end if
                end if

                if item 15 of propertiesToColumnsRef is not 0 then --
index15_hotkey_trigger
                    set theItem to hotkey trigger of eachCue
                    if theItem is enabled then
                        set item (item 15 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants15_hotkey_trigger
                    else if theItem is disabled then
                        set item (item 15 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants15_hotkey_trigger
                    end if
                end if

                if item 16 of propertiesToColumnsRef is not 0 then --
index16_midi_trigger
                    set theItem to midi trigger of eachCue
                    if theItem is enabled then
                        set item (item 16 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants16_midi_trigger
                    else if theItem is disabled then
                        set item (item 16 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants16_midi_trigger
                    end if
                end if

                if item 17 of propertiesToColumnsRef is not 0 then --
index17_midi_command
                    set theItem to midi command of eachCue
                    if theItem is note_on then
                        set item (item 17 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants17_midi_command
                    else if theItem is note_off then
                        set item (item 17 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants17_midi_command
                    else if theItem is program_change then
                        set item (item 17 of propertiesToColumnsRef) of
thePropertiesRef to item 3 of constants17_midi_command
                    else if theItem is control_change then
                        set item (item 17 of propertiesToColumnsRef) of
thePropertiesRef to item 4 of constants17_midi_command
                    else if theItem is key_pressure then
                        set item (item 17 of propertiesToColumnsRef) of
thePropertiesRef to item 5 of constants17_midi_command
                    else if theItem is channel_pressure then
                        set item (item 17 of propertiesToColumnsRef) of
thePropertiesRef to item 6 of constants17_midi_command
                    end if
                end if
```

```
                if item 18 of propertiesToColumnsRef is not 0 then --
index18_midi_byte_one
                    set theItem to midi byte one of eachCue as text
                    set item (item 18 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 19 of propertiesToColumnsRef is not 0 then --
index19_midi_byte_two
                    if midi command of eachCue is not program_change and
midi command of eachCue is not channel_pressure then
                        set theItem to midi byte two of eachCue as text
                        set item (item 19 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                    end if
                end if

                if item 20 of propertiesToColumnsRef is not 0 then --
index20_midi_byte_one_string
                    set theItem to midi byte one string of eachCue as text
                    set item (item 20 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 21 of propertiesToColumnsRef is not 0 then --
index21_midi_byte_two_string
                    if midi command of eachCue is not program_change and
midi command of eachCue is not channel_pressure then
                        set theItem to midi byte two string of eachCue as
text
                        set item (item 21 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                    end if
                end if

                if item 22 of propertiesToColumnsRef is not 0 then --
index22_timecode_trigger
                    set theItem to timecode trigger of eachCue
                    if theItem is enabled then
                        set item (item 22 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants22_timecode_trigger
                    else if theItem is disabled then
                        set item (item 22 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants22_timecode_trigger
                    end if
                end if

                if item 23 of propertiesToColumnsRef is not 0 then --
index23_timecode_show_as_timecode
                    set theItem to timecode show as timecode of eachCue
```

```
                    if theItem is true then
                        set item (item 23 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants23_timecode_show_as_timecode
                    else if theItem is false then
                        set item (item 23 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants23_timecode_show_as_timecode
                    end if
                end if

                if item 24 of propertiesToColumnsRef is not 0 then --
index24_timecode_hours
                    set theItem to timecode hours of eachCue as text
                    set item (item 24 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 25 of propertiesToColumnsRef is not 0 then --
index25_timecode_minutes
                    set theItem to timecode minutes of eachCue as text
                    set item (item 25 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 26 of propertiesToColumnsRef is not 0 then --
index26_timecode_seconds
                    set theItem to timecode seconds of eachCue as text
                    set item (item 26 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 27 of propertiesToColumnsRef is not 0 then --
index27_timecode_frames
                    set theItem to timecode frames of eachCue as text
                    set item (item 27 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 28 of propertiesToColumnsRef is not 0 then --
index28_timecode_bits
                    set theItem to timecode bits of eachCue as text
                    set item (item 28 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 29 of propertiesToColumnsRef is not 0 then --
index29_wall_clock_trigger
                    set theItem to wall clock trigger of eachCue
                    if theItem is enabled then
                        set item (item 29 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants29_wall_clock_trigger
                    else if theItem is disabled then
```

```
                        set item (item 29 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants29_wall_clock_trigger
                    end if
                end if

                if item 30 of propertiesToColumnsRef is not 0 then --
index30_wall_clock_hours
                    set theItem to wall clock hours of eachCue as text
                    set item (item 30 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 31 of propertiesToColumnsRef is not 0 then --
index31_wall_clock_minutes
                    set theItem to wall clock minutes of eachCue as text
                    set item (item 31 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 32 of propertiesToColumnsRef is not 0 then --
index32_wall_clock_seconds
                    set theItem to wall clock seconds of eachCue as text
                    set item (item 32 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 33 of propertiesToColumnsRef is not 0 and theType is
in index33_mode then
                    set theItem to mode of eachCue
                    if theItem is cue_list then
                        set item (item 33 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants33_mode
                    else if theItem is fire_first_enter_group then
                        set item (item 33 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants33_mode
                    else if theItem is fire_first_go_to_next_cue then
                        set item (item 33 of propertiesToColumnsRef) of
thePropertiesRef to item 3 of constants33_mode
                    else if theItem is fire_all then
                        set item (item 33 of propertiesToColumnsRef) of
thePropertiesRef to item 4 of constants33_mode
                    else if theItem is fire_random then
                        set item (item 33 of propertiesToColumnsRef) of
thePropertiesRef to item 5 of constants33_mode
                    end if
                end if

                if item 34 of propertiesToColumnsRef is not 0 and theType is
in index34_sync_to_timecode then
                    set theItem to sync to timecode of eachCue
                    if theItem is enabled then
```

```
                    set item (item 34 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants34_sync_to_timecode
                else if theItem is disabled then
                    set item (item 34 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants34_sync_to_timecode
                end if
            end if

            if item 35 of propertiesToColumnsRef is not 0 and theType is
in index35_sync_mode then
                set theItem to sync mode of eachCue
                if theItem is mtc then
                    set item (item 35 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants35_sync_mode
                else if theItem is ltc then
                    set item (item 35 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants35_sync_mode
                end if
            end if

            if item 36 of propertiesToColumnsRef is not 0 and theType is
in index36_smpte_format then
                set theItem to smpte format of eachCue
                if theItem is fps_24 then
                    set item (item 36 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants36_smpte_format
                else if theItem is fps_25 then
                    set item (item 36 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants36_smpte_format
                else if theItem is fps_30_drop then
                    set item (item 36 of propertiesToColumnsRef) of
thePropertiesRef to item 3 of constants36_smpte_format
                else if theItem is fps_30_non_drop then
                    set item (item 36 of propertiesToColumnsRef) of
thePropertiesRef to item 4 of constants36_smpte_format
                end if
            end if

            if item 37 of propertiesToColumnsRef is not 0 and theType is
in index37_mtc_sync_source_name then
                set theItem to mtc sync source name of eachCue
                if theItem is missing value then
                    set item (item 37 of propertiesToColumnsRef) of
thePropertiesRef to userNoMTCSyncSource
                else
                    set item (item 37 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem as text)
                end if
            end if

            if item 38 of propertiesToColumnsRef is not 0 and theType is
```

```
in index38_ltc_sync_channel then
                    set theItem to ltc sync channel of eachCue as text
                    set item (item 38 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 39 of propertiesToColumnsRef is not 0 and theType is
in index39_patch then
                    set theItem to patch of eachCue as text
                    set item (item 39 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 40 of propertiesToColumnsRef is not 0 and theType is
in index40_start_time then
                    set theItem to start time of eachCue
                    set item (item 40 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSsss(theItem)
                end if

                if item 41 of propertiesToColumnsRef is not 0 and theType is
in index41_end_time then
                    set theItem to end time of eachCue
                    set item (item 41 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSsss(theItem)
                end if

                if item 42 of propertiesToColumnsRef is not 0 and theType is
in index42_play_count then
                    set theItem to play count of eachCue as text
                    set item (item 42 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 43 of propertiesToColumnsRef is not 0 and theType is
in index43_infinite_loop then
                    set theItem to infinite loop of eachCue
                    if theItem is true then
                        set item (item 43 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants43_infinite_loop
                    else if theItem is false then
                        set item (item 43 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants43_infinite_loop
                    end if
                end if

                if item 44 of propertiesToColumnsRef is not 0 and theType is
in index44_rate then
                    set theItem to rate of eachCue as text
                    set item (item 44 of propertiesToColumnsRef) of
thePropertiesRef to theItem
```

```
                end if

                if item 45 of propertiesToColumnsRef is not 0 and theType is
in index45_integrated_fade then
                    set theItem to integrated fade of eachCue
                    if theItem is enabled then
                        set item (item 45 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants45_integrated_fade
                    else if theItem is disabled then
                        set item (item 45 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants45_integrated_fade
                    end if
                end if

                if item 46 of propertiesToColumnsRef is not 0 and theType is
in index46_lock_fade_to_cue then
                    set theItem to lock fade to cue of eachCue
                    if theItem is enabled then
                        set item (item 46 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants46_lock_fade_to_cue
                    else if theItem is disabled then
                        set item (item 46 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants46_lock_fade_to_cue
                    end if
                end if

                if item 47 of propertiesToColumnsRef is not 0 and theType is
in index47_pitch_shift then
                    set theItem to pitch shift of eachCue
                    if theItem is enabled then
                        set item (item 47 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants47_pitch_shift
                    else if theItem is disabled then
                        set item (item 47 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants47_pitch_shift
                    end if
                end if

                if item 48 of propertiesToColumnsRef is not 0 and theType is
in index48_slice_markers then
                    set theItem to slice markers of eachCue
                    set item (item 48 of propertiesToColumnsRef) of
thePropertiesRef ¬
                        to my recordToDelimitedText(theItem,
userSliceRecordColumnDelimiter, userSliceRecordRowDelimiter)
                end if

                if item 49 of propertiesToColumnsRef is not 0 and theType is
in index49_last_slice_play_count then
                    set theItem to last slice play count of eachCue as text
                    set item (item 49 of propertiesToColumnsRef) of
```

```
thePropertiesRef to theItem
                end if

                if item 50 of propertiesToColumnsRef is not 0 and theType is
in index50_last_slice_infinite_loop then
                    set theItem to last slice infinite loop of eachCue
                    if theItem is true then
                        set item (item 50 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants50_last_slice_infinite_loop
                    else if theItem is false then
                        set item (item 50 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants50_last_slice_infinite_loop
                    end if
                end if

                if item 51 of propertiesToColumnsRef is not 0 and theType is
in index51_layer then
                    set theItem to layer of eachCue as text
                    if theItem is "1000" then
                        set theItem to userLayerThousandIsTop
                    else if theItem is "0" then
                        set theItem to userLayerZeroIsBottom
                    end if
                    set item (item 51 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 52 of propertiesToColumnsRef is not 0 and theType is
in index52_full_surface then
                    set theItem to full surface of eachCue
                    if theItem is true then
                        set item (item 52 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants52_full_surface
                    else if theItem is false then
                        set item (item 52 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants52_full_surface
                    end if
                end if

                if item 53 of propertiesToColumnsRef is not 0 and theType is
in index53_preserve_aspect_ratio then
                    set theItem to preserve aspect ratio of eachCue
                    if theItem is true then
                        set item (item 53 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants53_preserve_aspect_ratio
                    else if theItem is false then
                        set item (item 53 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants53_preserve_aspect_ratio
                    end if
                end if
```

```
                if item 54 of propertiesToColumnsRef is not 0 and theType is
in index54_opacity then
                    set theItem to opacity of eachCue
                    if userConvertOpacityToPercent is true then
                        set item (item 54 of propertiesToColumnsRef) of
thePropertiesRef to ((round (100 * theItem) rounding toward zero) as text) &
"%"
                    else
                        set item (item 54 of propertiesToColumnsRef) of
thePropertiesRef to theItem as text
                    end if
                end if

                if item 55 of propertiesToColumnsRef is not 0 and theType is
in index55_translation_x then
                    set theItem to translation x of eachCue as text
                    set item (item 55 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 56 of propertiesToColumnsRef is not 0 and theType is
in index56_translation_y then
                    set theItem to translation y of eachCue as text
                    set item (item 56 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 57 of propertiesToColumnsRef is not 0 and theType is
in index57_scale_x then
                    set theItem to scale x of eachCue as text
                    set item (item 57 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 58 of propertiesToColumnsRef is not 0 and theType is
in index58_scale_y then
                    set theItem to scale y of eachCue as text
                    set item (item 58 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 59 of propertiesToColumnsRef is not 0 and theType is
in index59_do_video_effect then
                    set theItem to do video effect of eachCue
                    if theItem is true then
                        set item (item 59 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants59_do_video_effect
                    else if theItem is false then
                        set item (item 59 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants59_do_video_effect
                    end if
```

```
                end if

                if item 60 of propertiesToColumnsRef is not 0 and theType is
in index60_custom_quartz_file then
                    set theItem to custom quartz file of eachCue
                    if theItem is missing value then
                        set targetTitle to userMissingQuartzFile
                    else
                        set targetTitle to POSIX path of theItem as text
                    end if
                    set item (item 60 of propertiesToColumnsRef) of
thePropertiesRef to targetTitle
                end if

                if item 61 of propertiesToColumnsRef is not 0 and theType is
in index61_hold_at_end then
                    set theItem to hold at end of eachCue
                    if theItem is true then
                        set item (item 61 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants61_hold_at_end
                    else if theItem is false then
                        set item (item 61 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants61_hold_at_end
                    end if
                end if

                if item 62 of propertiesToColumnsRef is not 0 and theType is
in index62_text then
                    set theItem to text of eachCue as text
                    set item (item 62 of propertiesToColumnsRef) of
thePropertiesRef to my noReturns(my noTabs(theItem))
                end if

                if item 63 of propertiesToColumnsRef is not 0 and theType is
in index63_text_format then
                    set theItem to text format of eachCue as list
                    set coercedRecords to {}
                    repeat with eachRecord in theItem
                        set end of coercedRecords to my
coerceTextFormatRecord(eachRecord)
                    end repeat
                    set item (item 63 of propertiesToColumnsRef) of
thePropertiesRef to my listToDelimitedText(coercedRecords,
userTextFormatDelimiter)
                end if

                if item 64 of propertiesToColumnsRef is not 0 and theType is
in index64_text_alignment then
                    set theItem to text alignment of eachCue as text
                    set item (item 64 of propertiesToColumnsRef) of
thePropertiesRef to theItem
```

```
                end if

                if item 65 of propertiesToColumnsRef is not 0 and theType is
in index65_camera_patch then
                    set theItem to camera patch of eachCue as text
                    set item (item 65 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 66 of propertiesToColumnsRef is not 0 and theType is
in index66_audio_fade_mode then
                    set theItem to audio fade mode of eachCue
                    if theItem is absolute then
                        set item (item 66 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants66_audio_fade_mode
                    else if theItem is relative then
                        set item (item 66 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants66_audio_fade_mode
                    end if
                end if

                if item 67 of propertiesToColumnsRef is not 0 and theType is
in index67_video_fade_mode then
                    set theItem to video fade mode of eachCue
                    if theItem is absolute then
                        set item (item 67 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants67_video_fade_mode
                    else if theItem is relative then
                        set item (item 67 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants67_video_fade_mode
                    end if
                end if

                if item 68 of propertiesToColumnsRef is not 0 and theType is
in index68_stop_target_when_done then
                    set theItem to stop target when done of eachCue
                    if theItem is true then
                        set item (item 68 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants68_stop_target_when_done
                    else if theItem is false then
                        set item (item 68 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants68_stop_target_when_done
                    end if
                end if

                if item 69 of propertiesToColumnsRef is not 0 and theType is
in index69_rotation_type then
                    set theItem to rotation type of eachCue
                    set item (item 69 of propertiesToColumnsRef) of
thePropertiesRef to my lookUpBespoke(theItem, translation69_rotation_type)
                end if
```

```
                if item 70 of propertiesToColumnsRef is not 0 and theType is
in index70_rotation then
                    set theItem to rotation of eachCue as text
                    set item (item 70 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 71 of propertiesToColumnsRef is not 0 and theType is
in index71_do_opacity then
                    set theItem to do opacity of eachCue
                    if theItem is true then
                        set item (item 71 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants71_do_opacity
                    else if theItem is false then
                        set item (item 71 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants71_do_opacity
                    end if
                end if

                if item 72 of propertiesToColumnsRef is not 0 and theType is
in index72_do_translation then
                    set theItem to do translation of eachCue
                    if theItem is true then
                        set item (item 72 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants72_do_translation
                    else if theItem is false then
                        set item (item 72 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants72_do_translation
                    end if
                end if

                if item 73 of propertiesToColumnsRef is not 0 and theType is
in index73_do_rotation then
                    set theItem to do rotation of eachCue
                    if theItem is true then
                        set item (item 73 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants73_do_rotation
                    else if theItem is false then
                        set item (item 73 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants73_do_rotation
                    end if
                end if

                if item 74 of propertiesToColumnsRef is not 0 and theType is
in index74_do_scale then
                    set theItem to do scale of eachCue
                    if theItem is true then
                        set item (item 74 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants74_do_scale
                    else if theItem is false then
```

```
                        set item (item 74 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants74_do_scale
                    end if
                end if

                if item 75 of propertiesToColumnsRef is not 0 and theType is
in index75_command_text then
                    set theItem to command text of eachCue as text
                    set item (item 75 of propertiesToColumnsRef) of
thePropertiesRef to my noReturns(my noTabs(theItem))
                end if

                if item 76 of propertiesToColumnsRef is not 0 and theType is
in index76_always_collate then
                    set theItem to always collate of eachCue
                    if theItem is true then
                        set item (item 76 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants76_always_collate
                    else if theItem is false then
                        set item (item 76 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants76_always_collate
                    end if
                end if

                if item 77 of propertiesToColumnsRef is not 0 and theType is
in index77_message_type then
                    set theItem to message type of eachCue
                    if theItem is voice then
                        set item (item 77 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants77_message_type
                    else if theItem is msc then
                        set item (item 77 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants77_message_type
                    else if theItem is sysex then
                        set item (item 77 of propertiesToColumnsRef) of
thePropertiesRef to item 3 of constants77_message_type
                    end if
                end if

                if item 78 of propertiesToColumnsRef is not 0 and theType is
in index78_command then
                    set theItem to command of eachCue
                    if theItem is note_on then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants78_command
                    else if theItem is note_off then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants78_command
                    else if theItem is program_change then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 3 of constants78_command
```

```
                    else if theItem is control_change then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 4 of constants78_command
                    else if theItem is key_pressure then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 5 of constants78_command
                    else if theItem is channel_pressure then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 6 of constants78_command
                    else if theItem is pitch_bend then
                        set item (item 78 of propertiesToColumnsRef) of
thePropertiesRef to item 7 of constants78_command
                    end if
                end if

                if item 79 of propertiesToColumnsRef is not 0 and theType is
in index79_channel then
                    set theItem to channel of eachCue as text
                    set item (item 79 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 80 of propertiesToColumnsRef is not 0 and theType is
in index80_byte_one then
                    if command of eachCue is not pitch_bend then
                        set theItem to byte one of eachCue as text
                        set item (item 80 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                    end if
                end if

                if item 81 of propertiesToColumnsRef is not 0 and theType is
in index81_byte_two then
                    if command of eachCue is not pitch_bend and command of
eachCue is not program_change and ¬
                        command of eachCue is not channel_pressure then
                        set theItem to byte two of eachCue as text
                        set item (item 81 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                    end if
                end if

                if item 82 of propertiesToColumnsRef is not 0 and theType is
in index82_byte_combo then
                    if command of eachCue is pitch_bend then
                        set theItem to ((byte combo of eachCue) - 8192) as
text -- Pitch bend of 0 in the Inspector is reported to AppleScript as 8192
                        set item (item 82 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                    end if
                end if
```

```
                if item 83 of propertiesToColumnsRef is not 0 and theType is
in index83_end_value then
                    if command of eachCue is not pitch_bend then
                        set theItem to end value of eachCue as text
                    else
                        set theItem to ((end value of eachCue) - 8192) as
text -- Pitch bend of 0 in the Inspector is reported to AppleScript as 8192
                    end if
                    set item (item 83 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 84 of propertiesToColumnsRef is not 0 and theType is
in index84_fade then
                    set theItem to fade of eachCue
                    if theItem is enabled then
                        set item (item 84 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants84_fade
                    else if theItem is disabled then
                        set item (item 84 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants84_fade
                    end if
                end if

                if item 85 of propertiesToColumnsRef is not 0 and theType is
in index85_deviceID then
                    set theItem to deviceID of eachCue as text
                    set item (item 85 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 86 of propertiesToColumnsRef is not 0 and theType is
in index86_command_format then
                    set theItem to command format of eachCue
                    set item (item 86 of propertiesToColumnsRef) of
thePropertiesRef to my lookUpBespoke(theItem, translation86_command_format)
                end if

                if item 87 of propertiesToColumnsRef is not 0 and theType is
in index87_command_number then
                    set theItem to command number of eachCue
                    set item (item 87 of propertiesToColumnsRef) of
thePropertiesRef to my lookUpBespoke(theItem, translation87_command_number)
                end if

                if item 88 of propertiesToColumnsRef is not 0 and theType is
in index88_q__number then
                    set theItem to q_number of eachCue as text
                    set item (item 88 of propertiesToColumnsRef) of
thePropertiesRef to theItem
```

```
                end if

                if item 89 of propertiesToColumnsRef is not 0 and theType is
in index89_q__list then
                    set theItem to q_list of eachCue as text
                    set item (item 89 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 90 of propertiesToColumnsRef is not 0 and theType is
in index90_q__path then
                    set theItem to q_path of eachCue as text
                    set item (item 90 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 91 of propertiesToColumnsRef is not 0 and theType is
in index91_macro then
                    set theItem to macro of eachCue as text
                    set item (item 91 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 92 of propertiesToColumnsRef is not 0 and theType is
in index92_control_number then
                    set theItem to control number of eachCue as text
                    set item (item 92 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 93 of propertiesToColumnsRef is not 0 and theType is
in index93_control_value then
                    set theItem to control value of eachCue as text
                    set item (item 93 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 94 of propertiesToColumnsRef is not 0 and theType is
in index94_hours then
                    set theItem to hours of eachCue as text
                    set item (item 94 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 95 of propertiesToColumnsRef is not 0 and theType is
in index95_minutes then
                    set theItem to minutes of eachCue as text
                    set item (item 95 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if
```

```
                if item 96 of propertiesToColumnsRef is not 0 and theType is
in index96_seconds then
                    set theItem to seconds of eachCue as text
                    set item (item 96 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 97 of propertiesToColumnsRef is not 0 and theType is
in index97_frames then
                    set theItem to frames of eachCue as text
                    set item (item 97 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 98 of propertiesToColumnsRef is not 0 and theType is
in index98_subframes then
                    set theItem to subframes of eachCue as text
                    set item (item 98 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 99 of propertiesToColumnsRef is not 0 and theType is
in index99_send_time_with_set then
                    set theItem to send time with set of eachCue
                    if theItem is true then
                        set item (item 99 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants99_send_time_with_set
                    else if theItem is false then
                        set item (item 99 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants99_send_time_with_set
                    end if
                end if

                if item 100 of propertiesToColumnsRef is not 0 and theType
is in index100_sysex_message then
                    set theItem to sysex message of eachCue as text
                    set item (item 100 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 101 of propertiesToColumnsRef is not 0 and theType
is in index101_osc_message_type then
                    set theItem to osc message type of eachCue
                    if theItem is qlab then
                        set item (item 101 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants101_osc_message_type
                    else if theItem is custom then
                        set item (item 101 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants101_osc_message_type
                    else if theItem is udp then
                        set item (item 101 of propertiesToColumnsRef) of
```

```
thePropertiesRef to item 3 of constants101_osc_message_type
                    end if
                end if

                if item 102 of propertiesToColumnsRef is not 0 and theType
is in index102_q_num then
                    set theItem to q_num of eachCue as text
                    set item (item 102 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 103 of propertiesToColumnsRef is not 0 and theType
is in index103_q_command then
                    set theItem to q_command of eachCue
                    set item (item 103 of propertiesToColumnsRef) of
thePropertiesRef to my lookUpBespoke(theItem, translation103_q_command)
                end if

                if item 104 of propertiesToColumnsRef is not 0 and theType
is in index104_q_params then
                    set theItem to q_params of eachCue as text
                    set item (item 104 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 105 of propertiesToColumnsRef is not 0 and theType
is in index105_custom_message then
                    set theItem to custom message of eachCue as text
                    set item (item 105 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 106 of propertiesToColumnsRef is not 0 and theType
is in index106_udp_message then
                    set theItem to udp message of eachCue as text
                    set item (item 106 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 107 of propertiesToColumnsRef is not 0 and theType
is in index107_start_time_offset then
                    -- ###FIXME### It would be nice to format this as
HH:MM:SS:FF or HH:MM:SS;FF, but smpte format property broken in 4.1.6
                    set theItem to start time offset of eachCue as text
                    set item (item 107 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 108 of propertiesToColumnsRef is not 0 and theType
is in index108_fire_next_cue_when_slice_ends then
                    set theItem to fire next cue when slice ends of eachCue
```

```
                    if theItem is true then
                        set item (item 108 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants108_fire_next_cue_when_slice_ends
                    else if theItem is false then
                        set item (item 108 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants108_fire_next_cue_when_slice_ends
                    end if
                end if

                if item 109 of propertiesToColumnsRef is not 0 and theType
is in index109_stop_target_when_slice_ends then
                    set theItem to stop target when slice ends of eachCue
                    if theItem is true then
                        set item (item 109 of propertiesToColumnsRef) of
thePropertiesRef to item 1 of constants109_stop_target_when_slice_ends
                    else if theItem is false then
                        set item (item 109 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants109_stop_target_when_slice_ends
                    end if
                end if

                if item 110 of propertiesToColumnsRef is not 0 and theType
is in index110_load_time then
                    set theItem to load time of eachCue
                    set item (item 110 of propertiesToColumnsRef) of
thePropertiesRef to my makeHHMMSSsss(theItem)
                end if

                if item 111 of propertiesToColumnsRef is not 0 and theType
is in index111_assigned_number then
                    set theItem to assigned number of eachCue as text
                    set item (item 111 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 112 of propertiesToColumnsRef is not 0 and theType
is in index112_script_source then
                    set theItem to script source of eachCue as text
                    set item (item 112 of propertiesToColumnsRef) of
thePropertiesRef to my noReturns(my noTabs(theItem))
                end if

                if item 113 of propertiesToColumnsRef is not 0 then --
index113_parent
                    set cueListDetector to cue id "[root group of cue
lists]" -- Cue lists & carts have this as their parent
                    set theItem to parent of eachCue
                    if theItem is cueListDetector then -- Cue is a cue list
                        set targetTitle to userCueListsAreOrphans
                    else
                        set targetTitle to (q name of theItem) as text
```

```
                        if targetTitle is "" then
                            set targetTitle to (q number of theItem) as text
                            if targetTitle is "" then
                                set targetTitle to "id: " & (uniqueID of
theItem) as text
                            end if
                        end if
                    end if
                    if parent of theItem is cueListDetector then -- Parent
is a cue list
                        set targetTitle to item 1 of
userParentIsCueListBrackets & targetTitle & item 2 of
userParentIsCueListBrackets
                    end if
                    set item (item 113 of propertiesToColumnsRef) of
thePropertiesRef to targetTitle
                end if

                if item 114 of propertiesToColumnsRef is not 0 then --
index114_unique_ID
                    set theItem to uniqueID of eachCue as text
                    set item (item 114 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 115 of propertiesToColumnsRef is not 0 then --
index115_q_list_name
                    set theItem to q list name of eachCue as text
                    set item (item 115 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 116 of propertiesToColumnsRef is not 0 then --
index116_q_display_name
                    set theItem to q display name of eachCue as text
                    set item (item 116 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 117 of propertiesToColumnsRef is not 0 then --
index117_q_default_name
                    set theItem to q default name of eachCue as text
                    set item (item 117 of propertiesToColumnsRef) of
thePropertiesRef to my noTabs(theItem)
                end if

                if item 118 of propertiesToColumnsRef is not 0 then --
index118_broken
                    set theItem to broken of eachCue
                    if theItem is true then
                        set item (item 118 of propertiesToColumnsRef) of
```

```
thePropertiesRef to item 1 of constants118_broken
                    else if theItem is false then
                        set item (item 118 of propertiesToColumnsRef) of
thePropertiesRef to item 2 of constants118_broken
                    end if
                end if

                if item 119 of propertiesToColumnsRef is not 0 and theType
is in index119_audio_input_channels then
                    set theItem to audio input channels of eachCue as text
                    set item (item 119 of propertiesToColumnsRef) of
thePropertiesRef to theItem
                end if

                if item 120 of propertiesToColumnsRef is not 0 and theType
is in index120_start_value then
                    set theItem to end value of eachCue
                    if theItem is not missing value then
                        if command of eachCue is not pitch_bend then
                            set item (item 120 of propertiesToColumnsRef) of
thePropertiesRef to theItem as text
                        else
                            set item (item 120 of propertiesToColumnsRef) of
thePropertiesRef to (theItem - 8192) as text
                            -- Pitch bend of 0 in the Inspector is reported
to AppleScript as 8192
                        end if
                    end if
                end if

                if theType is in index_takesLevel then
                    if theType is not "Fade" then -- This bit stops the
script from reporting rows that aren't valid
                        set audioInputs to audio input channels of eachCue
                    else
                        if broken of eachCue is false then
                            try -- This protects against the cue target
being a Group Cue
                                set audioInputs to audio input channels of
cue target of eachCue
                            on error
                                set audioInputs to qLabMaxAudioInputs
                            end try
                        else
                            set audioInputs to qLabMaxAudioInputs
                        end if
                    end if
                    set AppleScript's text item delimiters to ","
                    repeat with eachLevelColumn in levelColumns
                        set crossPoint to item eachLevelColumn of
headerRowRef
```

```
                        set theRow to text item 1 of crossPoint as integer
                        if theRow > audioInputs then
                            set theLevel to userIrrelevantCrosspoints
                        else
                            set theColumn to text item 2 of crossPoint as
integer
                            set theLevel to eachCue getLevel row theRow
column theColumn as number
                            if theLevel ≤ item 1 of userMinusInfinity then
                                set theLevel to item 2 of userMinusInfinity
                            end if
                        end if
                        set item eachLevelColumn of thePropertiesRef to
theLevel
                    end repeat
                    set AppleScript's text item delimiters to tab
                end if

                if theType is in index_takesGang then
                    if theType is not "Fade" then -- This bit stops the
script from reporting rows that aren't valid
                        set audioInputs to audio input channels of eachCue
                    else
                        if broken of eachCue is false then
                            try -- This protects against the cue target
being a Group Cue
                                set audioInputs to audio input channels of
cue target of eachCue
                            on error
                                set audioInputs to qLabMaxAudioInputs
                            end try
                        else
                            set audioInputs to qLabMaxAudioInputs
                        end if
                    end if
                    set AppleScript's text item delimiters to "^"
                    repeat with eachGangColumn in gangColumns
                        set crossPoint to item eachGangColumn of
headerRowRef
                        set theRow to text item 1 of crossPoint as integer
                        if theRow > audioInputs then
                            set theGang to userIrrelevantCrosspoints
                        else
                            set theColumn to text item 2 of crossPoint as
integer
                            set theGang to eachCue getGang row theRow column
theColumn as text
                        end if
                        if theGang is not missing value then
                            set item eachGangColumn of thePropertiesRef to
theGang
```

```
                        end if
                    end repeat
                    set AppleScript's text item delimiters to tab
                end if

                -- Add thePropertiesRef to the end of theText

                set AppleScript's text item delimiters to tab
                set theText to theText & return & thePropertiesRef as text
                set AppleScript's text item delimiters to ""

                -- Countdown timer (and opportunity to escape)

                if i mod userEscapeHatchInterval is 0 and (countCues - i) >
userEscapeHatchInterval / 2 then -- Countdown timer (and opportunity to
escape)
                    tell me to countdownTimer(i, countCues, "cues") -- This
form avoids privileges error getting current date within an application tell
block
                end if

            end repeat

        end tell

    end tell

    -- Write the text back out to the file

    makeFileFromText(outputFile, theText)

    -- All done. Hoopla!

    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to makeNiceT(timeTaken)
    tell application id "com.figure53.QLab.4"
        activate
        set whatNext to button returned of (display dialog "Done." & return
& return & "(That took " & timeString & ".)" with title dialogTitle with
icon 1 ¬
            buttons {"Open in Excel", "Open in TextEdit", "Reveal in
Finder"} default button "Reveal in Finder" giving up after 60)
        (* ### NO EXCEL ### Replace the lines above with these lines if you
don't have Excel:
    set whatNext to button returned of (display dialog "Done." & return &
return & "(That took " & timeString & ".)" with title dialogTitle with icon
1 ¬
        buttons {"Open in TextEdit", "Reveal in Finder"} default button
"Reveal in Finder" giving up after 60)
        *)
```

```
    end tell

    set AppleScript's text item delimiters to currentTIDs

    if whatNext is "Open in Excel" then
        try -- ### NO EXCEL ### Delete this try block (^^^ to here ^^^) if
you don't have Excel
            tell application "Microsoft Excel"
                set openAllColumnsAsText to {}
                repeat with i from 1 to count headerRowRef
                    set end of openAllColumnsAsText to {i, text format}
                end repeat
                launch -- This protects against a very odd error message you
see if Excel wasn't running:
                (* "Sorry, we couldn't find
~/Library/Containers/com.microsoft.Excel/Data/name. Is it possible it was
moved, renamed or deleted?" *)
                open file outputFile -- ###FIXME### This seems to be the
only way of getting Microsoft Excel to allow "open text file" to work
                close workbook outputFilename
                activate
                open text file filename outputFile field info
openAllColumnsAsText origin Macintosh with tab
            end tell
        on error
            display dialog "That didn't work for some reason…" with title
dialogTitle with icon 0 buttons {"OK"} default button "OK"
        end try -- ### NO EXCEL ### ^^^ to here ^^^
    else if whatNext is "Open in TextEdit" then
        try
            tell application "TextEdit"
                activate
                open file outputFile
                set zoomed of front window to true
            end tell
        on error
            display dialog "That didn't work for some reason…" with title
dialogTitle with icon 0 buttons {"OK"} default button "OK"
        end try
    else
        tell application "Finder"
            activate
            reveal outputFile
        end tell
    end if

on error number -128
    set AppleScript's text item delimiters to currentTIDs
end try

-- Subroutines
```

```
(* === INPUT === *)

on enterANumberWithRangeWithCustomButton(thePrompt, defaultAnswer, ¬
    lowRange, acceptEqualsLowRange, highRange, acceptEqualsHighRange,
integerOnly, customButton, defaultButton) -- [Shared subroutine; minor mod]
    -- tell application id "com.figure53.QLab.4" [Minor mod]
    set theQuestion to ""
    repeat until theQuestion is not ""
        set {theQuestion, theButton} to {text returned, button returned} of
(display dialog thePrompt with title dialogTitle ¬
            default answer defaultAnswer buttons (customButton as list) &
{"Cancel", "OK"} default button defaultButton cancel button "Cancel")
        if theButton is customButton then
            set theAnswer to theButton
            exit repeat
        end if
        try
            if integerOnly is true then
                set theAnswer to theQuestion as integer -- Detects non-
numeric strings
                if theAnswer as text is not theQuestion then -- Detects non-
integer input
                    set theQuestion to ""
                end if
            else
                set theAnswer to theQuestion as number -- Detects non-
numeric strings
            end if
            if lowRange is not false then
                if acceptEqualsLowRange is true then
                    if theAnswer < lowRange then
                        set theQuestion to ""
                    end if
                else
                    if theAnswer ≤ lowRange then
                        set theQuestion to ""
                    end if
                end if
            end if
            if highRange is not false then
                if acceptEqualsHighRange is true then
                    if theAnswer > highRange then
                        set theQuestion to ""
                    end if
                else
                    if theAnswer ≥ highRange then
                        set theQuestion to ""
                    end if
                end if
            end if
```

```
        on error
            set theQuestion to ""
        end try
    end repeat
    return theAnswer
    -- end tell
end enterANumberWithRangeWithCustomButton

(* === OUTPUT === *)

on exitStrategy(theMessage, givingUp) -- [Shared subroutine]
    if theMessage is not false then
        if givingUp is not false then
            display dialog theMessage with title dialogTitle with icon 0
buttons {"OK"} default button "OK" giving up after givingUp
        else
            display dialog theMessage with title dialogTitle with icon 0
buttons {"OK"} default button "OK"
        end if
    end if
    set AppleScript's text item delimiters to currentTIDs
end exitStrategy

on startTheClock() -- [Shared subroutine]
    tell application id "com.figure53.QLab.4"
        display dialog "One moment caller…" with title dialogTitle with icon
1 buttons {"OK"} default button "OK" giving up after 1
    end tell
    set startTime to current date
end startTheClock

on countdownTimer(thisStep, totalSteps, whichCuesString) -- [Shared
subroutine]
    set timeTaken to round (current date) - startTime rounding as taught in
school
    set timeString to my makeMSS(timeTaken)
    tell application id "com.figure53.QLab.4"
        if frontmost then
            display dialog "Time elapsed: " & timeString & " — " & thisStep
& " of " & totalSteps & " " & whichCuesString & ¬
                " done…" with title dialogTitle with icon 1 buttons
{"Cancel", "OK"} default button "OK" cancel button "Cancel" giving up after
1
        end if
    end tell
end countdownTimer

(* === TIME === *)

on makeHHMMSSss(howLong) -- [Shared subroutine]
    set howManyHours to howLong div 3600
```

```
    if howManyHours is 0 then
        set hourString to ""
    else
        set hourString to my padNumber(howManyHours, 2) & ":"
    end if
    set howManyMinutes to (howLong mod 3600) div 60
    set minuteString to my padNumber(howManyMinutes, 2)
    set howManySeconds to howLong mod 60 div 1
    set secondString to my padNumber(howManySeconds, 2)
    set howManyFractionalSeconds to howLong mod 1
    set howManyRoundedSeconds to round 100 * howManyFractionalSeconds
rounding as taught in school
    set fractionString to my padNumber(howManyRoundedSeconds, 2)
    return hourString & minuteString & ":" & secondString & "." &
fractionString
end makeHHMMSSss

on makeHHMMSSsss(howLong) -- [Shared subroutine]
    set howManyHours to howLong div 3600
    if howManyHours is 0 then
        set hourString to ""
    else
        set hourString to my padNumber(howManyHours, 2) & ":"
    end if
    set howManyMinutes to (howLong mod 3600) div 60
    set minuteString to my padNumber(howManyMinutes, 2)
    set howManySeconds to howLong mod 60 div 1
    set secondString to my padNumber(howManySeconds, 2)
    set howManyFractionalSeconds to howLong mod 1
    set howManyRoundedSeconds to round 1000 * howManyFractionalSeconds
rounding as taught in school
    set fractionString to my padNumber(howManyRoundedSeconds, 3)
    return hourString & minuteString & ":" & secondString & "." &
fractionString
end makeHHMMSSsss

on makeMSS(howLong) -- [Shared subroutine]
    set howManyMinutes to howLong div 60
    set howManySeconds to howLong mod 60 div 1
    return (howManyMinutes as text) & ":" & my padNumber(howManySeconds, 2)
end makeMSS

on makeNiceT(howLong) -- [Shared subroutine]
    if howLong < 1 then
        return "less than a second"
    end if
    set howManyHours to howLong div 3600
    if howManyHours is 0 then
        set hourString to ""
    else if howManyHours is 1 then
        set hourString to "1 hour"
```

```
    else
        set hourString to (howManyHours as text) & " hours"
    end if
    set howManyMinutes to howLong mod 3600 div 60
    if howManyMinutes is 0 then
        set minuteString to ""
    else if howManyMinutes is 1 then
        set minuteString to "1 minute"
    else
        set minuteString to (howManyMinutes as text) & " minutes"
    end if
    set howManySeconds to howLong mod 60 div 1
    if howManySeconds is 0 then
        set secondString to ""
    else if howManySeconds is 1 then
        set secondString to "1 second"
    else
        set secondString to (howManySeconds as text) & " seconds"
    end if
    set theAmpersand to ""
    if hourString is not "" then
        if minuteString is not "" and secondString is not "" then
            set theAmpersand to ", "
        else if minuteString is not "" or secondString is not "" then
            set theAmpersand to " and "
        end if
    end if
    set theOtherAmpersand to ""
    if minuteString is not "" and secondString is not "" then
        set theOtherAmpersand to " and "
    end if
    return hourString & theAmpersand & minuteString & theOtherAmpersand &
secondString
end makeNiceT

(* === DATA WRANGLING === *)

on lookUpBespoke(lookUpValue, lookUpTable)
    if lookUpValue is not in lookUpTable then
        return lookUpValue & " (no match)"
    end if
    repeat with i from 1 to (count lookUpTable) by 2
        if lookUpValue is item i of lookUpTable then
            return item (i + 1) of lookUpTable
        end if
    end repeat
end lookUpBespoke

(* === TEXT WRANGLING === *)

on listToDelimitedText(theList, theDelimiter) -- [Shared subroutine]
```

```
    set passedTIDs to AppleScript's text item delimiters
    set AppleScript's text item delimiters to theDelimiter
    set delimitedText to theList as text
    set AppleScript's text item delimiters to passedTIDs
    return delimitedText
end listToDelimitedText

on padNumber(theNumber, minimumDigits) -- [Shared subroutine]
    set paddedNumber to theNumber as text
    repeat while (count paddedNumber) < minimumDigits
        set paddedNumber to "0" & paddedNumber
    end repeat
    return paddedNumber
end padNumber

on recordToDelimitedText(theRecord, theColumnDelimiter, theRowDelimiter) --
[Shared subroutine]
    set passedTIDs to AppleScript's text item delimiters
    set delimitedList to {}
    set AppleScript's text item delimiters to theColumnDelimiter
    repeat with eachItem in theRecord
        set end of delimitedList to (eachItem as list) as text
    end repeat
    set AppleScript's text item delimiters to theRowDelimiter
    set delimitedText to delimitedList as text
    set AppleScript's text item delimiters to passedTIDs
    return delimitedText
end recordToDelimitedText

on sortTextIgnoringCase(theText) -- [Shared subroutine]
    return do shell script "echo " & quoted form of theText & " | sort -f "
end sortTextIgnoringCase

on noTabs(dirtyText)
    set passedTIDs to AppleScript's text item delimiters
    set AppleScript's text item delimiters to tab
    set dirtyList to text items of dirtyText
    set AppleScript's text item delimiters to userTabCharacterSubstitute
    set cleanText to dirtyList as text
    set AppleScript's text item delimiters to passedTIDs
    return cleanText
end noTabs

on noReturns(dirtyText)
    set passedTIDs to AppleScript's text item delimiters
    set dirtyParas to paragraphs of dirtyText
    set AppleScript's text item delimiters to userCarriageReturnsInLongText
    set cleanText to dirtyParas as text
    set AppleScript's text item delimiters to passedTIDs
    return cleanText
end noReturns
```

```
on coerceTextFormatRecord(theRecord) -- ###FIXME### No customisation
implemented; ugly solution!
    tell application id "com.figure53.QLab.4" -- Needs to be inside tell
block so that record labels are understood (they're in QLab's sdef)
        set passedTIDs to AppleScript's text item delimiters
        set formatList to {"rangeOffset:" & rangeOffset of range of
theRecord, ¬
            "rangeLength:" & rangeLength of range of theRecord, ¬
            "fontFamily:" & fontFamily of theRecord, ¬
            "fontName:" & fontName of theRecord, ¬
            "fontSize:" & fontSize of theRecord, ¬
            "fontStyle:" & fontStyle of theRecord, ¬
            "red:" & red of rgbaColor of theRecord, ¬
            "alpha:" & alpha of rgbaColor of theRecord, ¬
            "blue:" & blue of rgbaColor of theRecord, ¬
            "green:" & green of rgbaColor of theRecord, ¬
            "lineSpacing:" & lineSpacing of theRecord}
        set AppleScript's text item delimiters to "; "
        set formatText to ("{" & formatList as text) & "}"
        set AppleScript's text item delimiters to passedTIDs
        return formatText
    end tell
end coerceTextFormatRecord

(* === FILES === *)

on makeFileFromText(outputFilePath, fileContents) -- [Shared subroutine]
    copy (open for access outputFilePath with write permission) to
theOpenFile
    set eof theOpenFile to 0 -- Clear it out first (just in case it already
existed)
    write fileContents to theOpenFile
    close access theOpenFile
end makeFileFromText

on offerToSave(theMessage, theButtons, defaultButton, cancelButton,
saveButton, textToSave, theAppendix)
    if button returned of (display dialog theMessage with title dialogTitle
with icon 1 ¬
        buttons theButtons default button defaultButton cancel button
cancelButton) is saveButton then
        if userFileTimestampFormat is false then
            set fileTimeString to ""
        else
            set fileTimeString to " | " & my grabTimeForFilenameBespoke()
        end if
        set savedFile to ((path to desktop) as text) & "QLab | " &
dialogTitle & " | " & theAppendix & fileTimeString & ".txt"
        my checkForFile(savedFile)
        my makeFileFromText(savedFile, textToSave)
```

```
        tell application "Finder"
            reveal savedFile
            activate
        end tell
        delay 1
        activate
    end if
end offerToSave

on grabTimeForFilenameBespoke()
    return do shell script "date " & quoted form of userFileTimestampFormat
end grabTimeForFilenameBespoke

on checkForFile(theFile)
    tell application "System Events"
        set fileExists to exists file theFile
    end tell
    if fileExists is true then
        display dialog "The output file already exists…" with title
dialogTitle with icon 1 ¬
            buttons {"Abort", "Overwrite"} default button "Overwrite" cancel
button "Abort"
    end if
end checkForFile

(* END: Make a text file from cues *)
```