

Final Year Project:

MULTIMEDIA AND DIGITAL SYSTEMS

Cross-platform Media Centre:

Appendices (Program Code)

- Simon Metcalfe • Informatics • Tutorial Group: 9 •
- Project Supervisor: Dr. Natalia Beloff •

Appendices

No.	Page	Name	Description
1	2	Director Scripts	Media centre scripts.
2	77	Flash ActionScript	Code related to Flash GUI objects.
3	89	Activity Log	A log of work performed and events throughout the project.

Director Scripts

Movie Scripts

main

```

--main
--loads when the movie starts

global fileMgr
global config
global dbMgr
global flMgr
global plAudio
global plVideo
global backHistory --used to tell frames who the previous frame was
global fileTypes
global locations

on startMovie()
  --create instances of parent scripts
  config = new(script "CONFIG_MANAGER")
  fileMgr = new(script "FILE_MANAGER")
  dbMgr = new(script "DB_MANAGER")
  flMgr = new(script "FL_MANAGER")
  plAudio = new(script "PL_MANAGER", true)
  plVideo = new(script "PL_MANAGER", false)
  flashClick("", "") --clears mouse click buffer
  setRedirect("")
  startOfMovie() --session manager's things to run at startup
  --set up locations
  locations = [ \
    "cdRom": "F:\", \
    "memoryCard": "J:\", \
    "filmFolder": "C:\Media\Films\","\
    "musicVideoFolder": "C:\Media\Music Videos\","\
    "episodeFolder": "C:\Media\Episodes\","\
    "pictureFolder": "C:\Media\Pictures\" ]
  --types of file extension compatible with the media centre
  fileTypes = [ \
    "all": ["mp3", "wav", "wma", "avi", "mpg", "mpeg", "bmp", "gif", "jpeg", "jpg", "txt", "nfo", "m3u"], \
    "music": ["mp3", "wav", "wma"], \
    "video": ["avi", "mpg", "mpeg"], \
    "image": ["bmp", "gif", "jpg", "jpeg"], \
    "text": ["txt", "nfo"], \
    "playlist": ["m3u"] ]
  --self-maintaining backHistory array
  backHistory = [:]
  repeat with i = 1 to count(the markerList)
    backHistory.addProp((the markerList[i]), "")
  end repeat
  temp = fileMgr.tagRead(the moviePath&"silence.mp3") --triggers BinaryIO register nag at start
end

on stopMovie()
  --clear parent scripts/xtras initialised
  config = 0
  fileMgr = 0
  dbMgr = 0
  backHistory = 0
  fileTypes = 0
  --flMgr = 0 --code is run from this after it is cleared
  --clear the image thumbnails
  repeat with i = 1 to 9
    member("slot"&i&"Image").filename = "thumb.bmp"
    member("slot"&i&"Text").text = ""
  end repeat
  member("albumArt").filename = "thumb.bmp"
  member("slide").filename = "thumb.bmp"
end

--Returns ceiling of number
on ceil(no)
return integer(no + 0.4999999)
end

```

sessionManager

```

--session manager manages data appropriate to the session
--the data does not need to be saved, any saving will be done by configManager

global fileMgr
global config
global dbMgr
global flMgr
global backHistory

global fxObj

global MMStructure --the menu layout

global repeatMode --playlist repeat mode 0:off, 1:all, 2:track only
global currentNavType

global browserFileType
global currentPath
global currentPicturePath

--slide show global vars to set-up a show
global slideshowMode
global showArray
global showPath
global currentImage

on startOfMovie() --global stuff to be set up on movie start
  fxObj = new (xtra "FileXtra4") --this is global as used in many places
  initMainMenu()
  config.load()
  config.cfgList["folderRecursive"] = true
  --config.save()
  dbMgr.openDB()
  lastFlashEvent = ""
  repeatMode = 0
end

--Predefined message box settings
--these are used to display standard instances of the
--msg box such as for errors. this is so only the
--error needs to be passed, icons, titles etc are done here

on errorBox(errorTxt)
  flMgr.showMsgBox("Program Error",errorTxt, "error",["OK"],1)
end

--menu/start file browser methods

on gotoMenu(history)
  backHistory["mainmenu"] = history
  _movie.go(_movie.label("mainmenu"))
end

on gotoNowPlaying(history)
  backHistory["sound"] = history
  _movie.go(_movie.label("sound"))
end

on browseLibrary(navType, history)
  if not voidP(history) then
    if not voidP(navType) then
      currentNavType = navType
      backHistory["library"] = history
      _movie.go(_movie.label("library"))
    end if
  end if
end

on browse(location, fType, history)
  if not voidP(location) then
    if not voidP(fType) then
      if not voidP(history) then
        browserFileType = fType
        currentPath = location
        backHistory["browser"] = history
        _movie.go(_movie.label("browser"))
      end if
    end if
  end if
end

on browsePicture(location, history)
  if not voidP(location) then
    if not voidP(history) then
      currentPicturePath = location
      backHistory["picture"] = history
      _movie.go(_movie.label("picture"))
    end if
  end if
end

```

```

--slide show methods
on playSlideshow(shPath, shArray, history)
  if not voidP(shArray) then
    if not voidP(shPath) then
      if not voidP(history) then
        slideshowMode = true
        showPath = shPath
        showArray = shArray.duplicate()
        currentImage = 1
        backHistory["slideshow"] = history
        _movie.go(_movie.label("slideshow"))
      end if
    end if
  end if
end

on viewImage(fPath, fName, history)
  if not voidP(fName) then
    if not voidP(fPath) then
      if not voidP(history) then
        slideshowMode = false
        currentImage = fPath & fName
        backHistory["slideshow"] = history
        _movie.go(_movie.label("slideshow"))
      end if
    end if
  end if
end

on initMainMenu()
  MMStructure = \
  ["CD/DVD": [[\
    "Play CD/DVD":["playcd", 0],\
    "View all":["viewcdall", 0],\
    "View music":["viewcdmusic", 0],\
    "View videos":["viewcdvideos", 0],\
    "View pictures":["viewcdpictures", 0]\
  ], 1], \
  "Music": [[\
    "Artists": ["artists", 0], \
    "Albums": ["albums", 0], \
    "Genres": ["genres", 0], \
    "Songs": ["songs", 0], \
    "Playlists": ["playlists", 0]\
  ], 0], \
  "Videos": [[\
    "Films": ["films", 0], \
    "TV episodes": ["episode", 0], \
    "Music videos": ["musicvideo", 0]\
  ], 0], \
  "Pictures": [[\
    "My Pictures": ["browsepicture", 0], \
    "Browse folder": ["browsepicturefolder", 0], \
    "Slide shows": ["pictureslideshows", 0]\
  ], 0], \
  "Memory card": [[\
    "View all": ["viewmcall", 0], \
    "View music": ["viewmcmusic", 0], \
    "View videos": ["viewmcvideos", 0], \
    "View pictures": ["viewmcpictures", 0]\
  ], 0], \
  "Settings": [[\
    "Appearance": ["appearancesettings", 0], \
    "Music": ["musicsettings", 0], \
    "Video": ["videosettings", 0], \
    "Pictures": ["picturesettings", 0], \
    "System": ["systemsettings", 0]\
  ], 0]]
end

--DB creation
on addMedia()
  folderSelected = fileMgr.openFolderWindow("Choose your MP3's folder")
  if folderSelected = "" then
    alert("You must select a folder to search for MP3 files!")
  else
    put folderSelected
    dbMgr.updateDB([folderSelected]) --usage updateDB([path1,path2,etc...])
  end if
end

```

playManager

```

--playManager

global fileMgr
global plAudio
global plVideo

global currentAudioPlayer --sound/wmp/real/qt

```

```

global currentAudioPlayMode --0=nofile,1=stop, 2=play,3=pause
global currentAudioFile
global currentAudioFileTags

global currentVideoPlayer
global currentVideoPlayMode
global currentVideoFile

on initVideoPlayers()
  member("wmpVideo").fileName = the moviePath & "blank.avi"
  member("realVideo").fileName = the moviePath & "blank.avi"
  member("qtVideo").fileName = the moviePath & "blank.avi"
  currentVideoPlayMode = 0
  currentVideoPlayer = ""
  currentVideoFile = ""
end

on initMediaPlayers()
  --ensures no players have files loaded/are playing at startup
  silence = the moviePath & "silence.wav"
  member("wmpAudio").fileName = silence
  member("realAudio").fileName = silence
  member("qtAudio").fileName = silence
end

on stopMediaPlayers()
  --occurs separately so there is no small clip of last file played upon start-up
  currentAudioPlayMode = 0
  currentAudioPlayer = ""
  currentAudioFile = ""
  _movie.sprite["wmpAudioInst"].stop()
  _movie.sprite["realAudioInst"].stop()
  _movie.sprite["qtAudioInst"].movieRate = 0
end

on audioTicker()
  doUpdate = false
  if currentAudioPlayMode = 2 then
    --if playing
    if getCurrentTime() >= getDuration() then
      --if end of song reached
      doUpdate = true
      plAudio.advance()
    end if
  end if
  return doUpdate
end

--enqueue methods (automatically create array element required by PL_MANAGER)

--avMode: use true to add to audio playlist, false for video
--id optional: if audio from db then it can store the id entry

on queueUp(avMode, id, fName)
  if not voidP(avMode) then
    if not voidP(fName) then
      if avMode = true then
        --add to audio pl
        theid = 0
        if not voidP(id) then
          theid = id
        end if
        --if fileMgr.getExtension(fName) = "mp3" then
        --mp3 file - get tag info if possible - this may be slow with a folder
        --tagData = fileMgr.tagRead(fName)
        --thename = tagData["title"]
        --plAudio.queueTrack([thename, theid, fName, 0, 0, 0])
        --else
        --use filename instead
        thename = fileMgr.getNameNoExt(fName)
        plAudio.queueTrack([thename, theid, fName, 0, 0, 0])
        --end if
      else
        --add to video pl
        thename = fileMgr.getNameNoExt(fName)
        plVideo.queueTrack([thename, 0, fName, 0, 0, 0])
      end if
    end if
  end if
end

on playAudio()
  if currentAudioPlayMode = 0 then
    --do nothing - perhaps a message box
  else if currentAudioPlayMode = 1 then
    --from stop to play
    currentAudioPlayMode = 2
    case (currentAudioPlayer) of
      "sound": sound(1).playFile(currentAudioFile)
      "wmp": _movie.sprite["wmpAudioInst"].play()
      "real": _movie.sprite["realAudioInst"].play()
      "qt": member("qtAudio").fileName = currentAudioFile
    end case
  else if currentAudioPlayMode = 3 then

```

```

--from pause to play
currentAudioPlayMode = 2
case (currentAudioPlayer) of
  "sound": sound(1).play()
  "wmp": _movie.sprite["wmpAudioInst"].play()
  "real": _movie.sprite["realAudioInst"].play()
  "qt": _movie.sprite["qtAudioInst"].movieRate = 1
end case
else if currentAudioPlayMode = 2 then
--from play to pause
currentAudioPlayMode = 3
case (currentAudioPlayer) of
  "sound": sound(1).pause()
  "wmp": _movie.sprite["wmpAudioInst"].pause()
  "real": _movie.sprite["realAudioInst"].pause()
  "qt": _movie.sprite["qtAudioInst"].movieRate = 0
end case
end if
end

on stopAudio()
  if currentAudioPlayMode = 2 or currentAudioPlayMode = 3 then
    currentAudioPlayMode = 1
    case (currentAudioPlayer) of
      "sound": sound(1).stop()
      "wmp": _movie.sprite["wmpAudioInst"].stop()
      "real": _movie.sprite["realAudioInst"].stop()
      "qt": _movie.sprite["qtAudioInst"].currentTime = 0 --make it go to the front of song
      _movie.sprite["qtAudioInst"].movieRate = 0
    end case
  end if
end

on ffAudio()
  if currentAudioPlayMode = 2 then
    case (currentAudioPlayer) of
      "sound": if (getCurrentTime() + 10000) < getDuration() then sound(1).currentTime =
sound(1).currentTime + 10000
      "wmp": _movie.sprite["wmpAudioInst"].currentTime = _movie.sprite["wmpAudioInst"].currentTime +
10000
      "real": _movie.sprite["realAudioInst"].currentTime = _movie.sprite["realAudioInst"].currentTime +
10000
      "qt": _movie.sprite["qtAudioInst"].currentTime = _movie.sprite["qtAudioInst"].currentTime + 10000
    end case
  end if
end

on rewAudio()
  if currentAudioPlayMode = 2 then
    case (currentAudioPlayer) of
      "sound": sound(1).currentTime = sound(1).currentTime - 10000
      "wmp": _movie.sprite["wmpAudioInst"].currentTime = _movie.sprite["wmpAudioInst"].currentTime -
10000
      "real": _movie.sprite["realAudioInst"].currentTime = _movie.sprite["realAudioInst"].currentTime -
10000
      "qt": _movie.sprite["qtAudioInst"].currentTime = _movie.sprite["qtAudioInst"].currentTime - 10000
    end case
  end if
end

on seekTo(ms)
  if not voidP(ms) then
    if currentAudioPlayMode = 2 then
      case (currentAudioPlayer) of
        "sound": sound(1).currentTime = ms
        "wmp": _movie.sprite["wmpAudioInst"].currentTime = ms
        "real": _movie.sprite["realAudioInst"].currentTime = ms
        "qt": _movie.sprite["qtAudioInst"].currentTime = ms
      end case
    end if
  end if
end

on playPercent(playMS,totalMS)
  if playMS > 0 and totalMS > 0 then
    return ((playMS + 0.0) / (totalMS + 0.0)) * 100
  else
    return 0
  end if
end

on msecToHMS(ms)
  --converts ms to HH:MM:SS to nicely display time
  --derived from 'generic mp3 player.dir'
  sec = integer((ms)/1000) -- get sseconds from milliseconds
  minutes= integer(sec/60) -- get minutes from seconds
  sec = sec - minutes*60 -- remove extra seconds so less than 60
  hour = integer(minutes/60) -- get hours from minutes, though not used in this timer.
  minutes = minutes - hour*60 -- etc.
  -- convert numbers to strings to be displayed
  hourString = string(hour+100).char[2..3]&":"
  minString = string(minutes+100).char[2..3]&":"
  secString = string(sec+100).char[2..3]
  if hourString = "00:" then hourString = ""
  timeDisplay=hourString&minString&secString -- sets variable to time string in minutes and seconds.
  return timeDisplay --returns time string

```

```

end

on msToTicks(ms)
  return (ms / 1000.0) * 60.0
end

on ticksToMs(tick)
  return (tick / 60.0) * 1000.0
end

on getCurrentTime()
  if currentAudioPlayMode = 2 then
    case (currentAudioPlayer) of
      "sound": return integer(sound(1).currentTime)
      "wmp":   return integer(_movie.sprite["wmpAudioInst"].currentTime)
      "real":  return integer(_movie.sprite["realAudioInst"].currentTime)
      "qt":    return integer(_movie.sprite["qtAudioInst"].currentTime)
    end case
  else
    return 0
  end if
end

on getDuration()
  if currentAudioPlayMode = 2 then
    case (currentAudioPlayer) of
      "sound": return integer(sound(1).endTime)
      "wmp":   return integer(_movie.sprite["wmpAudioInst"].duration)
      "real":  return integer(_movie.sprite["realAudioInst"].duration)
      "qt":    return integer(ticksToMs(_movie.sprite["qtAudioInst"].duration))
    end case
  else
    return 0
  end if
end

on playAudioFile(fn)
  stopAudio()
  currentAudioPlayer = ""
  currentAudioFile = fn
  if fileMgr.searchList(["mp3", "wav", "aif", "aiff"], fileMgr.getExtension(fn)) = true then
    currentAudioPlayer = "sound"
  else if fileMgr.searchList(["wma", "aac", "ogg"], fileMgr.getExtension(fn)) = true then
    currentAudioPlayer = "wmp"
  else if fileMgr.searchList(["ram", "ra", "rm"], fileMgr.getExtension(fn)) = true then
    currentAudioPlayer = "real"
  else if fileMgr.searchList(["mp4", "m4a"], fileMgr.getExtension(fn)) = true then
    currentAudioPlayer = "qt"
  else
    put errorBox("Audio file " & fn & " unsupported.")
    currentAudioPlayer = ""
    currentAudioPlayMode = 0
    currentAudioFile = ""
    exit
  end if
  put "Chosen audio player: " & currentAudioPlayer
  currentAudioFileTags = void
  if fileMgr.getExtension(fn) = "mp3" then
    --read id3 tags (before engaging playback)
    currentAudioFileTags = fileMgr.tagRead(fn)
  end if
  if currentAudioPlayer <> "" then
    currentAudioPlayMode = 2
    case (currentAudioPlayer) of
      "sound": sound(1).playFile(fn)
      "wmp":   member("wmpAudio").fileName = fn
      "real":  member("realAudio").fileName = fn
      "qt":    member("qtAudio").fileName = fn
    end case
  end if
end

on playVideoFile(fn)
  stopAudio()
  currentVideoPlayer = ""
  currentVideoFile = fn
  if fileMgr.searchList(["avi", "mpg", "mpeg", "wmv"], fileMgr.getExtension(fn)) = true then
    currentVideoPlayer = "wmp"
  else if fileMgr.searchList(["ram", "ra", "rm"], fileMgr.getExtension(fn)) = true then
    currentVideoPlayer = "real"
  else if fileMgr.searchList(["mov", "mp4"], fileMgr.getExtension(fn)) = true then
    currentVideoPlayer = "qt"
  else
    put errorBox("Video file " & fn & " unsupported.")
    currentVideoPlayer = ""
    currentVideoPlayMode = 0
    currentVideoFile = ""
    exit
  end if
  put "Chosen video player: " & currentVideoPlayer
  if currentVideoPlayer <> "" then
    currentVideoPlayMode = 1 --this is done upon video init
    case (currentVideoPlayer) of
      "wmp": _movie.go(_movie.label("wmpVid"))
      "real": movie.go(movie.label("realVid"))
    end case
  end if
end

```

```

    "qt": _movie.go(_movie.label("qtVid"))
  end case
end if
end

```

inputManager

```

--inputManager

--Manages remote>keyboard mapping.
--Each frame in movie uses on keyUp and on mouseUp which is run when a key is pressed.
--The frame code calls getKey() and getMouseClicked() to find out which key was pressed.

global flMgr

global lastFlashClick
global lastFlashEvent
global keyRedirect
global keyRedirectTo

on getKey
  case (the key) of
    -- power on
    "~": return "powerKey"
    -- d-pad
    "w": if keyRedirect = false then return "upKey"
         if keyRedirect = true then redirectKeys()
    "a": if keyRedirect = false then return "leftKey"
         if keyRedirect = true then redirectKeys()
    "s": if keyRedirect = false then return "downKey"
         if keyRedirect = true then redirectKeys()
    "d": if keyRedirect = false then return "rightKey"
         if keyRedirect = true then redirectKeys()
    "e": if keyRedirect = false then return "enterKey"
         if keyRedirect = true then redirectKeys()
    -- navigation
    "q": if keyRedirect = false then return "backKey"
         if keyRedirect = true then redirectKeys()
    "i": if keyRedirect = false then return "infoKey"
         if keyRedirect = true then redirectKeys()
    "h": if keyRedirect = false then return "helpKey"
         if keyRedirect = true then redirectKeys()
    "r": if keyRedirect = false then return "pageUpKey"
         if keyRedirect = true then redirectKeys()
    "f": if keyRedirect = false then return "pageDownKey"
         if keyRedirect = true then redirectKeys()
    "]": return "volUpKey"
    "[": return "volDownKey"
    -- number pad
    "1": return "1Key"
    "2": return "2Key"
    "3": return "3Key"
    "4": return "4Key"
    "5": return "5Key"
    "6": return "6Key"
    "7": return "7Key"
    "8": return "8Key"
    "9": return "9Key"
    "0": return "0Key"
    "*": return "wordKey"
    "-": return "delKey"
    -- play control
    "z": return "playKey"
    "x": return "nextKey"
    "c": return "prevKey"
    "v": return "stopKey"
    "\": return "nowPlayingKey"
    -- fast text
    "m": return "redKey"
    ",": return "greenKey"
    ".": return "yellowKey"
    "/": return "blueKey"
  end case
end

--RedirectKeys used to when a dialog box or fasttext menu is open
--and should have focus. Keystrokes are redirected to the procedures
--that control these items, and not to the frame.

--An IF statement is required for each item requiring redirection.
--It should list all keys that are temporarily redirected.
--keyRedirectTo should be the same as set by the procedure.

on redirectKeys()
  --redirect for ftMenu
  if keyRedirectTo = "FTMenu" then
    case (the key) of
      "e": flMgr.navFTMenu("enter") --tell current ft menu enter has been pressed
           return keyRedirectTo&"HasFocus"
      "w": flMgr.navFTMenu("up") --nav current ftmenu upwards
           return keyRedirectTo&"HasFocus"
      "s": flMgr.navFTMenu("down") --nav current ftmenu downwards
           return keyRedirectTo&"HasFocus"
    end case
  end
end

```

```

    "q": flMgr.navFTMenu("back") --tell current ftmenu to back out (clear FTmenufocus variable)
    return keyRedirectTo&"HasFocus"
  end case
end if
--redirect for textReader
if keyRedirectTo = "textReader" then
  case (the key) of
    "e": flMgr.navTextReader("close")
    return keyRedirectTo&"HasFocus"
    "w": flMgr.navTextReader("up")
    return keyRedirectTo&"HasFocus"
    "s": flMgr.navTextReader("down")
    return keyRedirectTo&"HasFocus"
    "q": flMgr.navTextReader("close")
    return keyRedirectTo&"HasFocus"
    "r": flMgr.navTextReader("pageUp")
    return keyRedirectTo&"HasFocus"
    "f": flMgr.navTextReader("pageDown")
    return keyRedirectTo&"HasFocus"
    "a": flMgr.navTextReader("prev")
    return keyRedirectTo&"HasFocus"
    "d": flMgr.navTextReader("next")
    return keyRedirectTo&"HasFocus"
  end case
end if
--redirect for msgBox
if keyRedirectTo = "msgBox" then
  case (the key) of
    "e": flMgr.navMsgBox("enter") --tell current ft menu enter has been pressed
    return keyRedirectTo&"HasFocus"
    "q": flMgr.navMsgBox("close")
    return keyRedirectTo&"HasFocus"
    "a": flMgr.navMsgBox("left")
    return keyRedirectTo&"HasFocus"
    "d": flMgr.navMsgBox("right")
    return keyRedirectTo&"HasFocus"
  end case
end if
end

--Sets redirect of keys to a particular item.

on setRedirect(redirectTo)
  if redirectTo = "" then
    keyRedirect = false
  else
    keyRedirect = true
    keyRedirectTo = redirectTo
  end if
end

on setFTMenuFocus(truefalse)
  FTMenuFocus = truefalse
end

on flashClick(buttonParent, buttonName)
  lastFlashClick = [buttonParent, buttonName]
end

on getMouseClick
  lastFlClick = lastFlashClick
  lastFlashClick = ["", ""]
  if not voidP(lastFlClick) then
    return lastFlClick
  else
    return ["", ""]
  end if
end

--Selections from Flash items

on flashEvent(event)
  lastFlashEvent = event
end

on getFlashEvent()
  event = lastFlashEvent
  lastFlashEvent = ""
  return event
end

--Notes on getting the last key pressed.
-----
--trace the key --returns char (most reliable)
--trace the keyPressed
--trace string(charToNum(the keyPressed))
--trace (the keyCode) --conflicts with keys
--keyDown does not work unless movie playing
--keyUp only reliable with _key

```

FILE_MANAGER

```

--FILE_MANAGER

property mp3Parser
property myFile
global fxObj

on new (me)
  --sets up extras required
  mp3Parser = new(script "MP3_PARSER")
  myFile = new(xtra "fileio")
  return me
end

--Methods for dealing with filenames.

on getExtension(me, fn)
  if not voidP(fn) then
    save = the itemDelimiter
    the itemDelimiter = "."
    f = the last item of fn
    the itemDelimiter = save
    return f
  end if
end

on getNameNoExt(me, fn)
  if not voidP(fn) then
    save = the itemDelimiter
    the itemDelimiter = "."
    --gets all items of filename except the last
    --this is in case there is more than 2 items, i.e. the filename contains "."
    f = (me.getFileName(fn)).item[1..(me.getFileName(fn).item.count) - 1]
    the itemDelimiter = save
    return f
  end if
end

on getFileName(me, fn)
  if not voidP(fn) then
    save = the itemDelimiter
    the itemDelimiter = "\"
    f = the last item of fn
    the itemDelimiter = save
    return f
  end if
end

--gets the path to the file without the filename
on getPath(me, fn)
  if not voidP(fn) then
    save = the itemDelimiter
    the itemDelimiter = "\"
    f = fn.item[1..(fn.item.count - 1)] & "\"
    the itemDelimiter = save
    return f
  end if
end

on getUpOneLevel(me, fn)
  --returns the path but up one level
  if not voidP(fn) then
    save = the itemDelimiter
    the itemDelimiter = "\"
    f = fn.item[1..(fn.item.count - 2)] & "\"
    if f = "\\\" then f = fn --if unc path and reached the computer name then do no more
    the itemDelimiter = save
    return f
  end if
end

on getContainingFolder(me, fn)
  --returns name of the folder that the file is in
  if not voidP(fn) then
    save = the itemDelimiter
    the itemDelimiter = "\"
    f = fn.item[(fn.item.count - 1)]
    the itemDelimiter = save
    return f
  end if
end

on searchList(me, list, value)
  if not voidP(list) then
    if not voidP(value) then
      repeat with s in list
        if s = value then
          return true
        end if
      end repeat
    end if
  end repeat
end

```

```

    return false
  end if
end if
end
end

--Folder read procedure to load the contents of a folder into a property list.
--Calls itself to read subfolders, will accept array of acceptable file extensions.

on folderRead(me, folderString , recursiveMode, sortMode, extFilter)
  if not voidP(folderString) then
    if not voidP(recursiveMode) then
      if not voidP(sortMode) then
        filenames = []
        folderList = fxObj.fx_FolderToList(folderString)
        if not voidP(folderList) then
          repeat with f in folderList
            if f contains "\" then
              if recursiveMode = true then
                sub = me.folderRead(folderString & f, recursiveMode, sortMode, extFilter)
                if not voidP(sub) then
                  --loop to add all subfolder items to filename list
                  repeat with i = 1 to count(sub)
                    --if no filter then add the file anyway
                    filenames.add(sub[i]) --add folder contents
                  end repeat
                end if
              end if
            else
              if voidP(extFilter) then
                --if no filter then add the file anyway
                --add file path/name (if not a folder) to list, get modification number
                filenames.add([folderString,f,(fxObj.fx_FileGetModNumber(folderString & f))])
              else
                --if filter list then check extension matches
                if me.searchList(extFilter, me.getExtension(f)) = true then
                  filenames.add([folderString,f,(fxObj.fx_FileGetModNumber(folderString & f))])
                end if
              end if
            end if
          end repeat
        end if
      end if
      if sortMode = true then
        sort(filenames) --sorts the list if folderSort is enabled
      end if
    end if
  end if
  return filenames
end

--ID3v1, ID3v2 tag reader. Makes best use of both tags available, or uses filenames.
--Removes 'The' from front of artist names, and converts xx/xx track no.s to xx.
--Does not modify the tags.

on tagRead(me, filename)
  if not voidP(filename) then
    mp3Parser.setFile(filename)
    info = mp3Parser.getMp3Info()
    --for simplicity in the repeat loop, these are named the same as values in vlprops
    --this can't replace vlprops as this is a "property list"
    tags = [ \
"track": "1", \
"title": "", \
"artist": "(no artist)", \
"album": "(no album)", \
"year": "", \
"genre": "Unknown Genre", \
"comment": "" ]
    --filename is used as title at first, but replaced by tag data if available
    tags["title"] = me.getNameNoExt(string(filename)) --mp3 extension removed
    tags["artist"] = me.getContainingFolder(string(filename)) --folder name used as artist tag
    --id3v1 reading is done next, but tags are replaced by v2 tags if they are found
    vlprops = ["title", "artist", "album", "year", "comment", "track", "genre"]
    --string to go through each of the v1 props
    if not voidP(info["id3v1"]) then
      repeat with prop in vlprops
        if rtrim(string(info["id3v1"][prop])) = "" then
          --DO NOTHING if the tag field exists but is empty
        else
          tags[prop] = rtrim(string(info["id3v1"][prop]))
        end if
      end repeat
    end if

    --the array v2props contains all names of id3v2 props that can be retrieved by mp3Parser
    v2props = [ \
"PCNT": "play_counter", \
"TRCK": "track", \
"TRK": "track", \
"TENC": "encoded", \
"TNC": "encoded", \
"WXXX": "link", \
"TCOP": "copyright", \
"TOPE": "original artist", \

```

```

"TCOM": "composer",\
"COMM": "comment",\
"COM": "comment",\
"TYER": "year ",\
"TYE": "year",\
"TIT2": "title",\
"TT2": "title",\
"TRCK": "track",\
"TRK": "track",\
"TPE1": "artist",\
"TPL": "artist",\
"TALB": "album",\
"TAL": "album",\
"WOAF": "audio_URL",\
"WOAR": "artist_URL",\
"WCOM": "buy_URL",\
"USLT": "lyrics"]
--unlike id3v1 this checks to see if the property is void first as there is two possible names
--using v2props.getPropAt(i) gives you PCNT, TRCK etc
--using v2props[i] gives you the text name
if not voidP(info["id3v2"]) then
  repeat with i = 1 to count(v2props)
    if not voidP(info["id3v2"][v2props.getPropAt(i)]) then
      if rtrim(string(info["id3v2"][v2props.getPropAt(i)])) = "" then
        --do nothing (the "if not" statement did not work!)
      else
        tags[v2props[i]] = rtrim(string(info["id3v2"][v2props.getPropAt(i)]))
      end if
    end if
  end repeat
--genre tagging is separate as it is more complex
--checks both TCON then TCO strings for genre, then if genre starts with "(" genre retrieved from
list
if not voidP(info["id3v2"]["TCON"]) then genre = rtrim(string(info["id3v2"]["TCON"]))
else if not voidP(info["id3v2"]["TCO"]) then genre = rtrim(string(info["id3v2"]["TCO"]))
if not voidP(genre) then
  if genre starts "(" then
    n = integer(genre.char[2..offset(")",genre]-1])
    genre = mp3Parser.pGenreList[n+1]
  end if
end if
if not voidP(genre) then
  if genre = "" then
    --do nothing
  else
    tags["genre"] = genre --only replace v1 tag if v2 tag exists
  end if
end if
--convert xx/xx track nos to xx
if tags["track"] contains "/" then
  save = the itemDelimiter
  the itemDelimiter = "/"
  tags["track"] = tags["track"].item[1]
  the itemDelimiter = save
end if
--convert 'The Band' to 'Band, The'
if chars(tags["artist"],1,4) = "The " then
  tags["artist"] = chars(tags["artist"],5, tags["artist"].length) & ", The"
end if
end if
return tags --any v2 tag types not in "tags" are added automatically
end

--Gets as much info about a file as possible
--If MP3 it returns organised tag information
--Other files the name, size, duration etc will be returned

on getInfo(me, fname, currentDur)
  if not voidP(fname) then
    infoString = ""
    infoString = infoString & "Filename: " & me.GetFileName(fname) & numToChar(13)
    infoString = infoString & "In folder: " & me.getContainingFolder(fname) & numToChar(13)
    infoString = infoString & "Size: " & integer((fxObj.fx_FileGetSize(fname)/1000))&"KB" & numToChar(13)
    infoString = infoString & "Modified: " & fxObj.fx_FileGetModDate(fname) --mode date contains its own
  ret char
  if currentDur = true then
    --duration can only be obtained if song is currently playing
    infoString = infoString & "Duration: " & msecToHMS(getCurrentTime()) & numToChar(13)
  end if
  if me.getExtension(fname) = "mp3" then
    tagData = me.tagRead(fname)
    infoString = infoString & numToChar(13) -- add a line between standard info and tag data
    infoString = infoString & "MP3 Tag Data" & numToChar(13) & "-----" & numToChar(13)
    repeat with i=1 to count(tagData)
      infoString = infoString & me.capSent(tagData.getPropAt(i))&": "& tagData[i] & numToChar(13)
    end repeat
  end if
  return infoString
end if
return ""
end

--Text file reading, loads file into variable.

```

```

on readTXTFile(me, fname)
  if not voidP(fname) then
    myFile.openFile(fname, 1) --open file read-only
    if myFile.status() <> 0 then
      put "FILEIO ERROR " & myFile.error(myFile.status())
      return "The file could not be read."
    else
      fileContents = myFile.readFile()
      myFile.closeFile()
      return fileContents
    end if
  end if
  return ""
end

--Text reader checks for txt file that matches media name, and others in folder.
--Uses text file reading procedure above.

on txtReader(me, fPath, fName)
  if not voidP(fPath) then
    if not voidP(fName) then
      match = ""
      others = []
      txtFiles = me.folderRead(fPath, false, false, ["txt", "nfo"])
      repeat with i = 1 to count(txtFiles)
        --check txt files names - do any match the filename
        if me.getNameNoExt(txtFiles[i][2]) = me.getNameNoExt(fName) then
          match = txtFiles[i][2]
        else
          others.add(txtFiles[i][2])
        end if
      end repeat
      --read the text files into an array
      txtFileData = []
      if match <> "" then txtFileData.add(me.readTXTFile(fPath & match))
      repeat with i = 1 to count(others)
        txtFileData.add(me.readTXTFile(fPath & others[i]))
      end repeat
      return txtFileData
    end if
  end if
  return []
end

--Art reader detects images in folder with music and classifies them.
--Detects largest album-art created by Windows Media Player, discards smaller image.

on artReader(me, fPath)
  if not voidP(fPath) then
    frontImg = ""
    backImg = ""
    cdImg = ""
    inlayImg = ""
    others = []
    imgFiles = me.folderRead(fPath, false, false, ["jpg", "jpeg", "bmp", "gif"])
    repeat with i = 1 to count(imgFiles)
      fn = imgFiles[i][2]
      --detect front cover
      if fn contains "AlbumArt" then
        if fn contains "large" then
          frontImg = fn
        end if
      else if fn contains "front" then
        frontImg = fn
      else if fn contains "folder" then
        frontImg = fn
      else if fn contains "cover" then
        frontImg = fn
      --detect back cover
      else if fn contains "back" then
        backImg = fn
      --detect cd scan
      else if fn contains "disc" then
        cdImg = fn
      else if fn contains "cd" then
        cdImg = fn
      --detect inside scan
      else if fn contains "inside" then
        inlayImg = fn
      else if fn contains "inlay" then
        inlayImg = fn
      --for art that is not classified
      else
        others.add(fn)
      end if
    end repeat
    art = []
    if frontImg <> "" then art.add(["Front Cover", frontImg])
    if backImg <> "" then art.add(["Back Cover", backImg])
    if inlayImg <> "" then art.add(["Inlay", inlayImg])
    if cdImg <> "" then art.add(["CD", cdImg])
    repeat with i = 1 to count(others)
      art.add(["Additional Art " & i, others[i]])
    end repeat
    return art
  end if
end

```

```

end if
end

--Playlist loading and saving.
--Checks basic and complex playlists (containing track name and time details).
--Detects different types of return characters used to denote new lines.

on loadM3U(me, fname)
  --loads m3u files into array: [time in seconds,printable name,filename],[time...
  myFile.openFile(fname, 1) --open file read-only
  if myFile.status() <> 0 then
    put "FILEIO ERROR " & myFile.error(myFile.status())
  else
    fileContents = myFile.readFile()
    myFile.closeFile()
    contentsArray = [] --file contents
    m3uFileData = [] --sorted m3u data
    save = the itemDelimiter
    --detect return character - most files contain 2 types of return char
    if fileContents contains RETURN then
      the itemDelimiter = RETURN
    else
      the itemDelimiter = numtochar(10)
    end if
    contentsArray.add(fileContents.item[1])
    if fileContents.item.count > 1 then
      repeat with i = 2 to fileContents.item.count
        contentsArray.add(me.rfc(fileContents.item[i]))
      end repeat
    end if
    --detect basic or complex m3u file
    if contentsArray[1] contains "#EXTM3U" then
      --type COMPLEX
      put "TYPE COMPLEX DETECTED"
      repeat with i = 2 to count(contentsArray)
        if (i mod 2) = 0 then --only if i is even
          if contentsArray[i] contains "#EXTINF" then
            if contentsArray[i+1] contains "#EXTINF" then
              --do nothing
            else
              temp = chars(contentsArray[i],9,contentsArray[i].length)
              the itemDelimiter = ","
              m3uFileData.add([temp.item[1],temp.item[2..(temp.item.count)],contentsArray[i+1]])
            end if
          end if
        end if
      end repeat
    else
      --type BASIC
      put "TYPE BASIC DETECTED"
      repeat with i = 1 to count(contentsArray)
        --should complete data from id3/filename here
        if contentsArray[i] <> "" then m3uFileData.add([0,"Unknown name",contentsArray[i]])
      end repeat
    end if
  end if
  put m3uFileData
  the itemDelimiter = save
end

on saveM3U(me, fpath, fname, songArray)
  --filename need not include extension - will be added if it doesnt
  --assumes song array [{"time","printable name" ,"path","filename"}]
  --always writes absolute paths in playlists
  if not voidP(fpath) then
    if not voidP(fname) then
      if not voidP(songArray) then
        if chars(fname,fname.length - 4,fname.length) = ".m3u" then
          saveFileName = fpath & fname
        else
          saveFileName = fpath & fname & ".m3u"
        end if
        myFile.createFile(saveFileName) --create file
        if myFile.status() <> 0 then put "FILEIO ERROR " & myFile.error(myFile.status())
        myFile.openFile(saveFileName, 2) --open file as write
        if myFile.status() <> 0 then put "FILEIO ERROR " & myFile.error(myFile.status())
      else
        --write the M3U file
        myFile.writeString("#EXTM3U")
        myFile.writeString(numToChar(13)&numToChar(10))
        repeat with i = 1 to count(songArray)
          myFile.writeString("#EXTINF:"& songArray[i][1] & "," & songArray[i][2])
          myFile.writeString(numToChar(13)&numToChar(10))
          myFile.writeString(songArray[i][3] & songArray[i][4])
          myFile.writeString(numToChar(13)&numToChar(10))
        end repeat
        myFile.closeFile()
      end if
    end if
  end if
end

--Detects volumes and controls of CD drive.

on openFolderWindow(me,windowTitle)

```

```

if not voidP(windowTitle) then
  folderSelected = string(fxObj.fx_FolderSelectDialog(windowTitle))
else
  folderSelected = string(fxObj.fx_FolderSelectDialog())
end if
if not voidP(folderSelected) then
  return folderSelected
else
  return ""
end if
end

on detectDrives(me)
  volumes = fxObj.fx_volumesToList()
  volArray = []
  repeat with i=1 to count(volumes)
    if fxObj.fx_volumeIsCDROM(volumes[i]) = 1 then
      volArray.addProp(volumes[i], "c")
    else
      if fxObj.fx_volumeIsRemovable(volumes[i]) then
        volArray.addProp(volumes[i], "r")
      else
        volArray.addProp(volumes[i], "f")
      end if
    end if
  end repeat
  return volArray
end

on ejectDrive(drive)
  if not voidP(drive) then
    fxObj.fx_volumeEject(drive)
  end if
end

-- utility methods -----
on rlc(me, str) --remove last character
  return chars(str,1,str.length - 1)
end

on rfc(me, str) --remove first character
  return chars(str,2,str.length)
end

-----
on capSent(me, str)
  -- converts the first letter of the first word of the str to uppercase
  -- str = forceLowerCase (Str)
  set lett = char 1 of str
  set tA = charToNum(lett)
  if tA = min(max(96, tA), 123) then
    set tA = tA - 32
    set lett = numTOChar(tA)
  end if
  set outPut = lett & char 2 to length(str) of str
  return output
end

-----
--Remove leading spaces
on rls(me, pStr)
  repeat while char 1 of pStr = " "
    delete char 1 of pStr
  end repeat
  return pStr
end

-----
--Remove trailing spaces
on rts(me, pStr)
  repeat while (the last char of pStr = " ")
    delete the last char of pStr
  end repeat
  return pStr
end

```

PL_MANAGER

```

--PL_MANAGER
--manages a playlist
--instances created for audio/video playlists
--isAudio must be set to determine playlist type (audio = true)

global repeatMode
global currentAudioPlayMode
global currentVideoPlayMode

property currentPlaylist
property isAudio --true for audio, false for video

on new(me, theType)
  currentPlaylist = []
  isAudio = theType
  return me
end

```

```

--for when the playlist is needed by an external script
on getPl(me)
  return currentPlaylist
end

on setPl(me,newPlaylist) --for when loading an m3u
  currentPlaylist = newPlaylist
end

on queueTrack(me,song)
  currentPlaylist.add(song)
  me.resetList()
end

on removeTrack(me,songNo) --probably never used
  currentPlaylist.deleteAt(songNo)
  me.resetList()
end

on clearPlaylist(me)
  --this should not be called directly but after a decision using:
  --flMgr.showMsgBox("Clear Playlist","Are you sure you want to clear the current
playlist?","question",["Yes","No"],2)
  currentPlaylist = []
  me.resetList()
  return true --same as doUpdate (always told to update)
end

--Used to move to and return the next song in the playlist
--Takes repeat mode into account plus play attributes
--Any songs flag not set to 'played' will be played before the playlist is declared complete

on advance(me)
  if count(currentPlaylist) > 0 then
    lastPlayed = 0
    nonePlayed = true
    nextSong = ""
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][6] = 2 then
        nonePlayed = false
      end if
      if currentPlaylist[n][6] = 1 then
        lastPlayed = n
        nonePlayed = false
        if repeatMode = 2 then
          --repeat last song played
          nextSong = currentPlaylist[n]
          exit repeat
        end if
      end if
    end repeat
    --set the 'played' flag on the last played song
    if lastPlayed > 0 then
      currentPlaylist[lastPlayed][6] = 2
    end if
    --if no songs have been played then play first
    if nonePlayed = true then
      currentPlaylist[1][6] = 1
      nextSong = currentPlaylist[1]
    end if
    --play the next file below last that hasnt been played
    if lastPlayed < count(currentPlaylist) and nextSong = "" then
      repeat with n = (lastPlayed+1) to count(currentPlaylist)
        if currentPlaylist[n][6] = 0 then
          currentPlaylist[n][6] = 1
          nextSong = currentPlaylist[n]
          exit repeat
        end if
      end repeat
    end if
    if lastPlayed > 1 and nextSong = "" then
      --check songs above last played
      repeat with n = 1 to (lastPlayed-1)
        if currentPlaylist[n][6] = 0 then
          currentPlaylist[n][6] = 1
          nextSong = currentPlaylist[n]
          exit repeat
        end if
      end repeat
    end if
    if nextSong = "" then
      --all songs have been played once
      clearPlayedAttrib()
      if repeatMode = 0 then
        --repeat off don't cycle playlist
        nextSong = "" --no next song, audio will be stopped
      else
        --repeat on: start playlist from beginning
        currentPlaylist[1][6] = 1
        nextSong = currentPlaylist[1]
      end if
    end if
    --play the chosen song if there is one
    if nextSong <> "" then

```

```

    if isAudio = true then
        playAudioFile(nextSong[3])
    else
        playVideoFile(nextSong[3])
    end if
else
    if isAudio = true then
        stopAudio()
    else
        --stopVideo()
    end if
end if
end if
end
end

--next/previous are simple and do not detect unplayed songs etc
--moving to songs does not affect their played attribute
--however it will be changed when the song completes and advance() is called

on nextSong(me)
doUpdate = false
if currentAudioPlayMode = 2 or currentVideoPlayMode = 2 then
    lastPlayed = 0
    if count(currentPlaylist) > 0 then
        repeat with n = 1 to count(currentPlaylist)
            if currentPlaylist[n][6] = 1 then
                lastPlayed = n
            end if
        end repeat
        if lastPlayed > 0 and lastPlayed < count(currentPlaylist) then
            doUpdate = true
            me.jumpTo(lastPlayed+1)
        end if
    end if
end if
return doUpdate
end

on prevSong(me)
doUpdate = false
if currentAudioPlayMode = 2 or currentVideoPlayMode = 2 then
    lastPlayed = 0
    if count(currentPlaylist) > 0 then
        repeat with n = 1 to count(currentPlaylist)
            if currentPlaylist[n][6] = 1 then
                lastPlayed = n
            end if
        end repeat
        if lastPlayed > 1 then
            doUpdate = true
            me.jumpTo(lastPlayed-1)
        end if
    end if
end if
return doUpdate
end

on jumpTo(me, trackNo)
if trackNo <= count(currentPlaylist) then
    repeat with n = 1 to count(currentPlaylist)
        --if other song has playing flag then clear it
        if currentPlaylist[n][6] = 1 then
            currentPlaylist[n][6] = 0
        end if
    end repeat
    currentPlaylist[trackNo][6] = 1
    if isAudio = true then
        playAudioFile(currentPlaylist[trackNo][3])
    else
        put currentPlaylist[trackNo][3]
        playVideoFile(currentPlaylist[trackNo][3])
    end if
end if
end

--navigation of selected item
--up/down/page up/page down

on navUp(me)
doUpdate = false
if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
        if currentPlaylist[n][4] = 1 then
            if n = 1 then
                exit repeat --at top do nothing
            else
                currentPlaylist[n][4] = 0
                currentPlaylist[n-1][4] = 1 --select item above
                doUpdate = true
                exit repeat
            end if
        end if
    end repeat
end if
end repeat

```

```

end if
return doUpdate
end

on navDown(me)
doUpdate = false
if count(currentPlaylist) > 0 then
repeat with n = 1 to count(currentPlaylist)
if currentPlaylist[n][4] = 1 then
if n = count(currentPlaylist) then
exit repeat -- at bottom do nothing
else
currentPlaylist[n][4] = 0
currentPlaylist[n+1][4] = 1 --select item below
doUpdate = true
exit repeat
end if
end if
end repeat
end if
return doUpdate
end

on navPageUp(me)
doUpdate = false
if count(currentPlaylist) > 0 then
repeat with n = 1 to count(currentPlaylist)
if currentPlaylist[n][4] = 1 then
if n = 1 then
exit repeat -- at top do nothing
else
if (n-16) < 1 then
--less than 16 from the top - go to top of list
currentPlaylist[n][4] = 0
currentPlaylist[1][4] = 1
--updateFlWindow()
else
--move 16 places up
currentPlaylist[n][4] = 0
currentPlaylist[n-16][4] = 1 --select item above
end if
doUpdate = true
exit repeat
end if
end if
end repeat
end if
return doUpdate
end

on navPageDown()
doUpdate = false
if count(currentPlaylist) > 0 then
repeat with n = 1 to count(currentPlaylist)
if currentPlaylist[n][4] = 1 then
if n = count(currentPlaylist) then
exit repeat -- at bottom do nothing
else
if (n+16) > count(currentPlaylist) then
--less than 16 items - go to bottom of list
currentPlaylist[n][4] = 0
currentPlaylist[count(currentPlaylist)][4] = 1
--updatePlWindow()
else
--move 16 places down
currentPlaylist[n][4] = 0
currentPlaylist[n+16][4] = 1 --select item above
end if
doUpdate = true
exit repeat
end if
end if
end repeat
end if
return doUpdate
end

--returns an items so it can be played
--(does not select if edit mode is on, sound script will call highlight() instead)

on navSelect(me)
selectedItem = ""
if count(currentPlaylist) > 0 then
repeat with n = 1 to count(currentPlaylist)
if currentPlaylist[n][4] = 1 then
selectedItem = currentPlaylist[n]
me.jumpTo(n)
--was currentPlaylist[n][1]
--tell to update
end if
end repeat
else
end if
return selectedItem
end

```

```

on navGetSelected()
  if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][4] = 1 then
        return currentPlaylist[n]
      end if
    end repeat
  return void
end if
end

on navGetPlaying()
  if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][6] = 1 then
        return currentPlaylist[n]
      end if
    end repeat
  return void
end if
end

--Delete the selected item or a group of selected items

on del(me)
  doUpdate = false
  --removes all selected songs
  --if no selected then removes current selection
  if count(currentPlaylist) > 0 then
    highlighted = false
    firstDelete = true
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][5] = 1 then
        highlighted = true
        if firstDelete = true then
          firstDelete = false
          currentPlaylist.deleteAt(n)
        else
          currentPlaylist.deleteAt(n)
        end if
      end if
    end repeat
    if highlighted = false then
      if me.deleteCurrent() = true then
        doUpdate = true
      end if
    else
      me.resetList() --list only reset if a selection is deleted
      doUpdate = true
    end if
  end if
  return doUpdate
end

on deleteCurrent()
  doUpdate = false
  --removes the song and selects another in the list
  if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][4] = 1 then
        if n < count(currentPlaylist) then
          currentPlaylist[n+1][4] = 1
        else if n > 1 then
          currentPlaylist[n-1][4] = 1
        else
          --all songs deleted
        end if
        currentPlaylist.deleteAt(n)
        doUpdate = true
      end if
    end repeat
  else
  end if
  return doUpdate
end

--Move single item or group of items

on moveUp(me)
  doUpdate = false
  if count(currentPlaylist) > 0 then
    highlighted = false
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][5] = 1 then
        highlighted = true
        if n = 1 then
          --at top of list cannot be moved up
          exit repeat
        else
          me.swapItems(n,n-1)
          doUpdate = true
        end if
      end if
    end repeat
  end if
end

```

```

    if highlighted = false then
      --move up currently selected file only
      repeat with n = 1 to count(currentPlaylist)
        if currentPlaylist[n][4] = 1 then
          if n = 1 then
            --at top cannot move up
          else
            me.swapItems(n,n-1)
            doUpdate = true
          end if
        end if
      end repeat
    end if
  end if
  return doUpdate
end

on moveDown(me)
  doUpdate = false
  if count(currentPlaylist) > 0 then
    highlighted = false
    repeat with n = count(currentPlaylist) down to 1
      if currentPlaylist[n][5] = 1 then
        highlighted = true
        if n = count(currentPlaylist) then
          --at bottom of list cannot be moved down
          exit repeat
        else
          me.swapItems(n,n+1)
          doUpdate = true
        end if
      end if
    end repeat
  end if
  if highlighted = false then
    --move down currently selected file only
    repeat with n = count(currentPlaylist) down to 1
      if currentPlaylist[n][4] = 1 then
        if n = count(currentPlaylist) then
          --at bottom cannot move up
        else
          me.swapItems(n,n+1)
          doUpdate = true
        end if
      end if
    end repeat
  end if
  return doUpdate
end

--Playlist sorting functions (randomise/alphabetical)

on randomise(me)
  doUpdate = false
  if count(currentPlaylist) > 1 then
    randomList = []
    repeat while count(currentPlaylist) > 0
      rnd = random(count(currentPlaylist))
      randomList.add(currentPlaylist[rnd])
      currentPlaylist.deleteAt(rnd)
    end repeat
    currentPlaylist = randomList
    me.resetList()
    doUpdate = true
  end if
  return doUpdate
end

on sortAlphabetical(me)
  doUpdate = false
  if count(currentPlaylist) > 1 then
    currentPlaylist.sort()
    me.resetList()
    doUpdate = true
  end if
  return doUpdate
end

--Creating selections

on highlight(me)
  doUpdate = false
  if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
      if currentPlaylist[n][4] = 1 then
        doUpdate = true
        if currentPlaylist[n][5] = 0 then
          --if not highlighted then highlight
          currentPlaylist[n][5] = 1
        else
          --if highlighted then unhighlight
          currentPlaylist[n][5] = 0
        end if
      end if
    end repeat
  end if
  --move to the next item if there is one

```

```

    if n = count(currentPlaylist) then
        exit repeat -- at bottom do nothing
    else
        currentPlaylist[n][4] = 0
        currentPlaylist[n+1][4] = 1 --select item below
        exit repeat
    end if
end if
end repeat
end if
return doUpdate
end

on highlightNone(me)
doUpdate = false
if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
        currentPlaylist[n][5] = 0
        doUpdate = true
    end repeat
end if
return doUpdate
end

on clearPlayedAttrib()
if count(currentPlaylist) > 0 then
    repeat with n = 1 to count(currentPlaylist)
        currentPlaylist[n][6] = 0
    end repeat
end if
end

--Internal Procedures Deselects
--swaps items in the list - used for shifting songs around

on resetList(me)
if count(currentPlaylist) > 0 then
    me.highlightNone()
    repeat with n = 1 to count(currentPlaylist)
        currentPlaylist[n][4] = 0
    end repeat
    currentPlaylist[1][4] = 1
end if
end

--Used to move items up and down a list
on swapItems (me,item1, item2)
i1 = currentPlaylist[item1]
i2 = currentPlaylist[item2]
currentPlaylist.setAt(item1,i2)
currentPlaylist.setAt(item2,i1)
end

```

DB_MANAGER

```

--DB MANAGER
--this script manages creating, populating, querying the database
--code for predicitive text queries is also found here

global fileMgr

property gDB
property DBFileName
property prMap
property selectionIDs

on new(me)
DBFileName = the moviepath & "mp3db_latest.adb" --DATABASE FILE PATH/NAME
gDB = new (xtra "arca")
if not objectp(gDB) then
    then errorBox("A database problem has occured. Arca DB instance not created.")
    return
end if
setPrMap()
selectionIDs = ["0":0,"1":0,"2":0,"3":0]
return me
end

on setPrMap(me)
--1 button acts as wild character (insted of 1/punctuation)
prMap = [ \
"1":[" "], \
"2":["a","b","c",2,"à","á","â","ã","ä","å","ç"], \
"3":["d","e","f",3,"è","é","ê","ë"], \
"4":["g","h","i",4,"ì","í","î"], \
"5":["j","k","l",5], \
"6":["m","n","o",6,"ñ","ò","ó","ô","õ","ø"], \
"7":["p","q","r","s",7], \
"8":["t","u","v",8,"û","ü","ý"], \
"9":["w","x","y","z",9,"ÿ"], \
"0":[" ",0]

```

```

end

--General DB initialisation, creating tables, deleting

on openDB(me)
  res = gDB.openDB(DBFileName)
  if res.errorMsg <> 0 then --not equal to 0
    if res.errorMsg = 604 then
      errorBox("The database does not exist, run createDB")
      --createDB
    else
      errorBox("A database error occured: "& gDB.explainError(res.errorMsg))
      return
    end if
  end if
end

on closeDB(me)
  gDB.closeDB()
end

on deleteDB(me)
  put "Database file " & DBFileName & " deleted!"
  --this does not work as db file appears to still be in use
  --resetdb() can be used instead
  gDB.closeDB()
  fdelete(DBFileName)
end

on resetDB(me)
  --delete tables/indexes
  gDB.executeSQL("DROP TABLE Artist")
  gDB.executeSQL("DROP TABLE Album")
  gDB.executeSQL("DROP TABLE Genre")
  gDB.executeSQL("DROP TABLE Path")
  gDB.executeSQL("DROP TABLE Song")
  gDB.executeSQL("DROP INDEX indexArtist")
  gDB.executeSQL("DROP INDEX indexAlbum")
  gDB.executeSQL("DROP INDEX indexGenre")
  --make them again
  gDB.executeSQL("CREATE TABLE Artist(ArtistID integer primary key, ArtistName text)")
  gDB.executeSQL("CREATE TABLE Album(AlbumID integer primary key, AlbumName text)")
  gDB.executeSQL("CREATE TABLE Genre(GenreID integer primary key, GenreName text)")
  gDB.executeSQL("CREATE TABLE Path(PathID integer primary key, PathStr text)")
  gDB.executeSQL("CREATE TABLE Song(SongID integer primary key, ArtistID integer, AlbumID integer, \
  GenreID integer, PathID integer, Filename text, ModNo numeric, Title text, TrackNo integer)")
  gDB.executeSQL("CREATE INDEX indexArtist ON Artist(ArtistName ASC)")
  gDB.executeSQL("CREATE INDEX indexAlbum ON Album(AlbumName ASC)")
  gDB.executeSQL("CREATE INDEX indexGenre ON Genre(GenreName ASC)")
  gDB.executeSQL("COMMIT")
end

on createDB(me)
  --make the DB (should check if it is there first)
  res = gDB.createDB(DBFileName)
  if res.errorMsg <> 0 then
    errorBox("A database error occured: "& gDB.explainError(res.errorMsg))
  else
    --create tables
    gDB.executeSQL("CREATE TABLE Artist(ArtistID integer primary key, ArtistName text)")
    gDB.executeSQL("CREATE TABLE Album(AlbumID integer primary key, AlbumName text)")
    gDB.executeSQL("CREATE TABLE Genre(GenreID integer primary key, GenreName text)")
    gDB.executeSQL("CREATE TABLE Path(PathID integer primary key, PathStr text)")
    gDB.executeSQL("CREATE TABLE Song(SongID integer primary key, ArtistID integer, AlbumID integer, \
    GenreID integer, PathID integer, Filename text, ModNo numeric, Title text, TrackNo integer)")
    gDB.executeSQL("CREATE INDEX indexArtist ON Artist(ArtistName ASC)")
    gDB.executeSQL("CREATE INDEX indexAlbum ON Album(AlbumName ASC)")
    gDB.executeSQL("CREATE INDEX indexGenre ON Genre(GenreName ASC)")
    gDB.executeSQL("COMMIT")
    --gDB.executeSQL("")
  end if
end

on commitDB(me, compact)
  --must do this after db changes
  gDB.executeSQL("COMMIT")
  if compact = true then
    trace "Compacting..."
    gdb.compactDB() --compact db
    trace "      ...done"
  end if
end

--Main procedure for creating the DB. Accepts array of folders.
--New items are added, changed items are updated (deleted-added).
--Obsolete items are removed.
--When a large folder is checked this is quite slow.

on updateDB(me, folders)
  if not voidP(folders) then
    put "Monitoring " & count(folders) & " folder(s)..."
    --read db contents (so all obsolete files known)
    put "Reading DB contents...please wait..."
    dbObsolete = me.getFolderPropList()
    --check each folder separately
  end if
end

```

```

repeat with m = 1 to count(folders)
  put "checking " & folders[m]
  folderContents = fileMgr.folderRead(folders[m], true, true, ["mp3"]) --folder/recursive mode/sort
mode/EXTENSION FILTER LIST
  put "...done. Comparing to database..."
  --check each file in dir for db presence
  repeat with f = 1 to count(folderContents)
    --get or make the folder ID
    pID = me.getID("Path", "PathID", "PathStr", folderContents[f][1])
    --check for file in song table
    dbResult = gDB.executeSQL("SELECT SongID, ModNo FROM Song WHERE PathID = ? AND Filename =
?", [pID, folderContents[f][2]])
    if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
    if count(dbResult.rows) = 0 then
      --file not in db - read tags and add it
      fileTags = fileMgr.tagRead(folderContents[f][1]&folderContents[f][2])
me.addTrack(fileTags["artist"], fileTags["album"], fileTags["genre"], folderContents[f][1], folderContents[f][
2], folderContents[f][3], fileTags["title"], fileTags["track"])
    else
      --file in db - has it changed?
      if dbResult.rows[1][2] = folderContents[f][3] then
        --file is up-to-date: remove from obsolete list
        dbObsolete.deleteProp(dbResult.rows[1][1])
      else
        --file changed - delete old, add new, remove from obsolete list
        put "A CHANGE DETECTED"
        put "OLD FILE MOD " & dbResult.rows[1][2]
        put "NEW FILE MOD " & folderContents[f][3]
        me.removeTrack(dbResult.rows[1][1])
        fileTags = fileMgr.tagRead(folderContents[f][1]&folderContents[f][2])
me.addTrack(fileTags["artist"], fileTags["album"], fileTags["genre"], folderContents[f][1], folderContents[f][
2], folderContents[f][3], fileTags["title"], fileTags["track"])
        dbObsolete.deleteProp(dbResult.rows[1][1])
        put "A file has been updated"
      end if
    end if
    --go on to next file
  end repeat
  --go on to next folder
end repeat
--now all folders have been checked, delete obsolete files
put "All folder reading done"
put count(dbObsolete) & " files to delete!"
repeat with d = 1 to count(dbObsolete)
  me.removeTrack(dbObsolete.getPropAt(d))
end repeat
end if
end
end

--Makes a property list from the DB the same as the folderRead method
--Used for updateDB
on getFolderPropList(me)
  dbResult = gDB.executeSQL("SELECT Song.SongID, Path.PathStr, Song.Filename, Song.ModNo FROM Path, Song
WHERE Path.PathID = Song.PathID")
  if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
  --convert to property list
  prList = []
  repeat with i = 1 to count(dbResult.rows)
    prList.addProp(dbResult.rows[i][1], [dbResult.rows[i][1], dbResult.rows[i][2], dbResult.rows[i][3]])
  end repeat
  return prList
end

--Procedures for adding/removing records
--All id3 data is passed but not all is written to the song table
--ids for art, alb, gen, path etc are looked up using procedures
on addTrack(me, artistTag, albumTag, genreTag, pathStr, fnStr, modStr, titleTag, trackNoTag)
  --get/make ids from artist, alb, gen, and path tables
  artID = me.getID("Artist", "ArtistID", "ArtistName", artistTag)
  albID = me.getID("Album", "AlbumID", "AlbumName", albumTag)
  genID = me.getID("Genre", "GenreID", "GenreName", genreTag)
  pathID = me.getID("Path", "PathID", "PathStr", pathStr)
  gDB.executeSQL("INSERT INTO Song (ArtistID, AlbumID, GenreID, PathID, Filename, ModNo, Title, TrackNo) \
VALUES (?, ?, ?, ?, ?, ?, ?, ?)", [artID, albID, genID, pathID, fnStr, modStr, titleTag, trackNoTag])
end

on removeTrack(me, trkID)
  --retrieve art/alb/gen/path ids
  dbResult = gDB.executeSQL("SELECT ArtistID, AlbumID, GenreID, PathID FROM Song WHERE SongID =
?", [trkID])
  if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
  if count(dbResult.rows) > 0 then
    artID = dbResult.rows[1][1]
    albID = dbResult.rows[1][2]
    genID = dbResult.rows[1][3]
    pathID = dbResult.rows[1][4]
    --delete song from song table
    me.deleteByID("Song", "SongID", trkID)
    --check for obsolete art/alb/gen/path and remove from tables as necessary

```

```

dbResult = gDB.executeSQL("SELECT SongID FROM Song WHERE ArtistID = ?", [artID])
if count(dbResult.rows) = 0 then me.deleteByID("Artist", "ArtistID", artID)
dbResult = gDB.executeSQL("SELECT SongID FROM Song WHERE AlbumID = ?", [albID])
if count(dbResult.rows) = 0 then me.deleteByID("Album", "AlbumID", albID)
dbResult = gDB.executeSQL("SELECT SongID FROM Song WHERE GenreID = ?", [genID])
if count(dbResult.rows) = 0 then me.deleteByID("Genre", "GenreID", genID)
dbResult = gDB.executeSQL("SELECT SongID FROM Song WHERE PathID = ?", [pathID])
if count(dbResult.rows) = 0 then me.deleteByID("Path", "PathID", pathID)
else --director bug - single line if statement cannot have another statement's 'else' under it
end if
else
put "ERROR - cannot delete song - it does not exist!"
end if
end

--Predictive search. Generates SQL statements for a string of numbers.
--Used in conjunction with library DB querying to find items in a list
--In practice, it is only used with temporary tables 0-4 but all modes work

on predictiveSearch(me, criteria, keyString)
--select the search mode - criteria accepts artist/album/genre/song/0/1/2/3 (no.s represent temp panel
tables)
srchTable = ""
srchField = ""
srchIDField = ""
case (criteria) of
"Artist":
srchTable = "Artist"
srchField = "ArtistName"
srchIDField = "ArtistID"
"Album":
srchTable = "Album"
srchField = "AlbumName"
srchIDField = "AlbumID"
"Genre":
srchTable = "Genre"
srchField = "GenreName"
srchIDField = "GenreID"
"Song":
srchTable = "Song"
srchField = "Title"
srchIDField = "SongID"
"0":
srchTable = "Panel0"
srchField = "TextStr"
srchIDField = "IdNo"
"1":
srchTable = "Panel1"
srchField = "TextStr"
srchIDField = "IdNo"
"2":
srchTable = "Panel2"
srchField = "TextStr"
srchIDField = "IdNo"
"3":
srchTable = "Panel3"
srchField = "TextStr"
srchIDField = "IdNo"
"All":
--do something special for all!
end case
--divide up string into array
keyArray = []
repeat with i = 1 to keyString.length
keyArray.add(chars(keyString, i, i))
end repeat
--check list contains at least 1 keypresses other than 1
keyCheck = keyArray.getOne("0")
repeat with i = 2 to 9
keyCheck = keyCheck + keyArray.getOne(string(i))
end repeat
if keyCheck = 0 then
return [] --return an empty array
exit
end if
--generate SQL
sqlString = "SELECT " & srchIDField & ", " & srchField & " FROM " & srchTable & " WHERE "
repeat with i = 1 to count(keyArray)
leadUscore = ""
if i > 1 then
sqlString = sqlString & " AND "
repeat with c = 2 to i --add _ to represent previous chars
leadUscore = leadUscore & "_"
end repeat
end if
sqlString = sqlString & "("
repeat with n = 1 to count(prMap[keyArray[i]])
if n > 1 then sqlString = sqlString & " OR "
sqlString = sqlString & srchField & " LIKE '" & leadUscore & prMap[keyArray[i]][n] & "%'"
end repeat
sqlString = sqlString & ")"
end repeat
sqlString = sqlString & " ORDER BY " & srchField --make return results in order
--execute SQL
dbResult = qDB.executeSQL(sqlString)

```

```

if dbResult.errorMessage <> 0 then errorBox("A database error occured (executing predictive SQL): "&
gDB.explainError(dbResult.errorMessage))
return dbResult.rows
end

--Library DB Querying (uses arca's createSelection to handle large volumes of results)
--Allows predefined queries, and the ability to get an IDNo to find an item in the DB.

on setPanel(me,panelNo,forTable,idFieldName, fieldName, refineIDField, refineIDValue)
--creates temp table that contains a row field to keep track of results
dbResult = gDB.executeSQL("DROP TABLE Panel"&panelNo) --deletes table in case it exists
dbResult = gDB.executeSQL("CREATE TEMPORARY TABLE Panel"&panelNo&"(RowNo integer primary key, IdNo
integer, TextStr text)")
if dbResult.errorMessage <> 0 then errorBox("A database error occured (creating temporary table): "&
gDB.explainError(dbResult.errorMessage))
dbResult = gDB.executeSQL("DROP VIEW ViewPanel"&panelNo) --delete view if it exists
if refineIDField = "" or refineIDValue = 0 then
--no refine - return all results
dbResult = gDB.executeSQL("CREATE VIEW ViewPanel"&panelNo&" AS SELECT " & idFieldName & ", " &
fieldName & " FROM " & forTable & " ORDER BY " & fieldName)
if dbResult.errorMessage <> 0 then errorBox("A database error occured (creating view): "&
gDB.explainError(dbResult.errorMessage))
--
else
--only return results that match the refine - slower but required
if forTable = "Song" then
--if listing songs that other tables need not be involved
sqlString = "CREATE VIEW ViewPanel"&panelNo&
" AS SELECT DISTINCT " & idFieldName & ", " & fieldName &
" FROM Song WHERE " & refineIDField & " = " & refineIDValue &
" ORDER BY " & fieldName
else
--results are from other tables (artist/album/genre)
sqlString = "CREATE VIEW ViewPanel"&panelNo&
" AS SELECT DISTINCT " & forTable & "." & idFieldName & ", " & forTable & "." & fieldName &
" FROM " & forTable & ", Song WHERE " & forTable & "." & idFieldName & " = Song." & idFieldName & \
" AND Song." & refineIDField & " = " & refineIDValue &
" ORDER BY " & forTable & "." & fieldName
end if
dbResult = gDB.executeSQL(sqlString)
if dbResult.errorMessage <> 0 then errorBox("A database error occured (creating view): "&
gDB.explainError(dbResult.errorMessage))
--
end if
dbResult = gDB.executeSQL("INSERT INTO Panel"&panelNo&"(IdNo, TextStr) SELECT * FROM ViewPanel"&panelNo)
if dbResult.errorMessage <> 0 then errorBox("A database error occured (populating temporary table): "&
gDB.explainError(dbResult.errorMessage))
if selectionIDs[panelNo] <> 0 then --check for existing selection, if exists remove to reclaim memory
--remove existing selection
gDB.freeSelection(selectionIDs[panelNo])
end if
dbSelection = gDB.createSelection("SELECT * FROM Panel"&panelNo)
if dbSelection.errorMessage <> 0 then errorBox("A database error occured (generating selection): "&
gDB.explainError(dbResult.errorMessage))
selectionIDs[panelNo] = dbSelection.selectionID
return dbSelection.numrows
end

on getPanelRowNo(me,panelNo,strText)
dbResult = gDB.executeSQL("SELECT RowNo FROM Panel"&panelNo&" WHERE TextStr LIKE ?",[strText&"%"])
if dbResult.errorMessage <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMessage))
else
if count(dbResult.rows) > 0 then
return dbResult.rows[1][1] --always returns the first result
else
return 0
end if
end if
end

on getPanelPage(me,panelNo,startRow,endRow)
dbResult = gDB.getRows(selectionIDs[panelNo],startRow,EndRow)
if dbResult.errorMessage <> 0 then
errorBox("A database error occured: "& gDB.explainError(dbResult.errorMessage))
return []
else
return dbResult.rows
end if
end

--Internal procedures for dealing with tables:
--Artist, album, genre, path.

--returns [artist,path,filename,track] for library
on getSong(me, id)
dbResult = []
if not voidP(id) then
dbResult = gDB.executeSQL("SELECT Artist.ArtistName, Path.PathStr, Song.FileName, Song.Title FROM
Artist,Path, Song WHERE Artist.ArtistID = Song.ArtistID AND Path.PathID = Song.PathID AND Song.SongID =
?",[id])
if dbResult.errorMessage <> 0 then
errorBox("A database error occured: "& gDB.explainError(dbResult.errorMessage))

```

```

    return []
  else
    return dbResult.rows
  end if
end if
end

on getField(me, forTable, fieldName)
  if not voidP(forTable) then
    if not voidP(fieldName) then
      sqlString = "SELECT " & fieldName & " FROM " & forTable
      dbResult = gDB.executeSQL(sqlString)
      if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
      return dbResult.rows
    end if
  end if
end

on deleteByID(me, forTable, idFieldName, idToDelete)
  put idFieldName && idToDelete & " REMOVED FROM " & forTable & " TABLE"
  sqlString = "DELETE FROM " & forTable & " WHERE " & idFieldName & " = ?"
  gDB.executeSQL(sqlString, [idToDelete])
end

on getRecordCount(me, forTable)
  dbResult = gDB.createSelection("SELECT ROWID FROM ?", [forTable])
  if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
  gDB.freeSelection(dbResult.selectionid) --free up memory
  return dbResult.numrows
end

on getID(me, forTable, idFieldName, fieldName, fieldString)
  --checks for art/alb etc already existing in the table
  --if it exists it returns the id, else an entry is made and the new id is returned
  --fieldString is passed to arca and not in SQL so escape chars are handled correcty
  sqlString = "SELECT " & idFieldName & " FROM " & forTable & " WHERE " & fieldName & " = ?"
  dbResult = gDB.executeSQL(sqlString, [fieldString])
  if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
  if dbResult.rows = [] then
    sqlString2 = "INSERT INTO " & forTable & " (" & fieldName & ") VALUES(?)"
    gDB.executeSQL(sqlString2, [fieldString])
    dbResult = gDB.executeSQL(sqlString, [fieldString])
    if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
  end if
  return dbResult.rows[1][1]
end

-----

--Experimental code for building dictionary (unused).
--Dictionary needs to remove all punctuation (esp from beginning and end of file).
--Must also remove numbers (or include them to ensure they can be typed as with letters).

on createDictionary(me)
  gDB.executeSQL("DROP TABLE Dictionary")
  gDB.executeSQL("CREATE TABLE Dictionary(WordID integer primary key, InArtist integer, InAlbum integer,
InGenre integer, InSong integer, WordStr text)")
  put "Dictionary Table Created"
  save = the itemDelimiter
  the itemDelimiter = " "
  put "Reading genres..."
  names = me.getField("Genre", "GenreName")
  put "...reading complete...writing to dictionary..."
  repeat with i = 1 to count(names)
    repeat with n = 1 to names[i][1].item.count
      me.addToDict(names[i][1].item[n], "Genre")
    end repeat
  end repeat
  put "Genres done..."
  put "Reading artists..."
  names = me.getField("Artist", "ArtistName")
  put "...reading complete...writing to dictionary..."
  repeat with i = 1 to count(names)
    repeat with n = 1 to names[i][1].item.count
      me.addToDict(names[i][1].item[n], "Artist")
    end repeat
  end repeat
  put "Artists done..."
  put "Reading albums..."
  names = me.getField("Album", "AlbumName")
  put "...reading complete...writing to dictionary..."
  repeat with i = 1 to count(names)
    repeat with n = 1 to names[i][1].item.count
      me.addToDict(names[i][1].item[n], "Album")
    end repeat
  end repeat
  put "Albums done..."
  the itemDelimiter = save
end

on addToDict(me, wordString, sourceTable)
  --adds words to dictionary-and set their source

```

```

sqlString = "SELECT WordID FROM Dictionary WHERE WordStr = ?"
dbResult = gDB.executeSQL(sqlString,[wordString])
if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
gDB.explainError(dbResult.errorMsg))
if dbResult.rows = [] then
  --if a new word
  artVal = 0
  albVal = 0
  genVal = 0
  songVal = 0
  case (sourceTable) of --say in which table the word was found
    "Artist": artVal = 1
    "Album": albVal = 1
    "Genre": genVal = 1
    "Song": songVal = 1
  end case
  sqlString2 = "INSERT INTO Dictionary (InArtist,InAlbum,InGenre,InSong,WordStr) VALUES(?,?,?,?)"
  gDB.executeSQL(sqlString2,[artVal,albVal,genVal,songVal,wordString])
  dbResult = gDB.executeSQL(sqlString,[wordString])
  if dbResult.errorMsg <> 0 then errorBox("A database error occured: "&
  gDB.explainError(dbResult.errorMsg))
  --put "Word " & dbResult.rows[1][1] && wordString " added"
else
  --if word already in dictioanry update in values
  gDB.executeSQL("UPDATE Dictionary SET In" &sourceTable& " = 1 WHERE WordID = ?",[dbResult.rows[1][1]])
  --put "Word " & dbResult.rows[1][1] && wordString " updated"
end if
end

```

FL_MANAGER

```

--FL_MANAGER

--Manages global flash objects:
--Fast text menu
--Universal dialog box
--Text reader window

property msgBoxButtonArray
property msgBoxCurrentButton
property msgBoxCurrentID

property textReaderArray
property textReaderCurrent --curent text displayed

property currentFTMenu
property currentFTMenuPos
property FTItems

on new(me)
  currentFTMenu = ""
  currentFTMenuPos = 0
  FTItems = ["red": [], "green": [], "yellow": [], "blue": []]
  return me
end

--Message box procedures

on initMsgBox(me)
  _movie.sprite["msgBoxInst"].visible = true
  _movie.sprite["msgBoxInst"].ink = 100
end

on showMsgBox(me,titleTxt,msgTxt,iconType,buttonArray,buttonDefault)
  if not voidP(titleTxt) then
    if not voidP(msgTxt) then
      if not voidP(iconType) then
        if not voidP(buttonArray) then
          if not voidP(buttonDefault) then
            msgBoxButtonArray = buttonArray
            msgBoxCurrentID = titleTxt --used to identify which msg box button is pressed
            _movie.sprite["msgBoxInst"].visible = true
            _movie.sprite["msgBoxInst"].ink = 100
            _movie.sprite["msgBoxInst"].setIcons(iconType)
            _movie.sprite["msgBoxInst"].textClip.textSetup(msgTxt)
            _movie.sprite["msgBoxInst"].title.text = titleTxt
            _movie.sprite["msgBoxInst"].setColours(0,75,255,100,0,75,255,100)
            if count(buttonArray) = 1 then _movie.sprite["msgBoxInst"].\
setButtons(_movie.sprite["msgBoxInst"].newObject("Array",buttonArray[1]))
            if count(buttonArray) = 2 then _movie.sprite["msgBoxInst"].\
setButtons(_movie.sprite["msgBoxInst"].newObject("Array",buttonArray[1],buttonArray[2]))
            if count(buttonArray) = 3 then _movie.sprite["msgBoxInst"].\
setButtons(_movie.sprite["msgBoxInst"].newObject("Array",buttonArray[1],buttonArray[2],buttonArray[3]))
            msgBoxCurrentButton = buttonDefault
            _movie.sprite["msgBoxInst"].selectButton(msgBoxCurrentButton - 1) --flash uses 0-2 for 1-3
            setRedirect("msgBox")
          end if
        end if
      end if
    end if
  end if
end if

```

```

end

on navMsgBox(me, nav)
  if nav = "close" then
    me.hideMsgBox()
  else if nav = "enter" then
    flashEvent ("msgBox_" & msgBoxCurrentID & "_" & msgBoxButtonArray[msgBoxCurrentButton])
    me.hideMsgBox()
  else if nav = "right" then
    if msgBoxCurrentButton < count(msgBoxButtonArray) then
      msgBoxCurrentButton = msgBoxCurrentButton + 1
      _movie.sprite["msgBoxInst"].selectButton(msgBoxCurrentButton - 1)
    end if
  else if nav = "left" then
    if msgBoxCurrentButton > 1 then
      msgBoxCurrentButton = msgBoxCurrentButton - 1
      _movie.sprite["msgBoxInst"].selectButton(msgBoxCurrentButton - 1)
    end if
  end if
end

on hideMsgBox(me)
  _movie.sprite["msgBoxInst"].visible = false
  _movie.sprite["msgBoxInst"].ink = 0 --setting visibility to 0 (opaque) should help reduce memory
  setRedirect("")
end

--Text file reader window procedures

on initTextReader(me)
  _movie.sprite["textWindowInst"].visible = true
  _movie.sprite["textWindowInst"].ink = 100
end

on showTextReader(me, titleTxt, textArray)
  if not voidP(titleTxt) then
    if not voidP(textArray) then
      textReaderArray = textArray
      textReaderCurrent = 1
      _movie.sprite["textWindowInst"].visible = true
      _movie.sprite["textWindowInst"].ink = 100
      _movie.sprite["textWindowInst"].setColours(0, 75, 255, 100, 0, 75, 255, 100)
      _movie.sprite["textWindowInst"].showLastButton("false")
      if count(textReaderArray) > 1 then
        _movie.sprite["textWindowInst"].showNextButton("true")
      else
        _movie.sprite["textWindowInst"].showNextButton("false")
      end if
      _movie.sprite["textWindowInst"].textClip.textSetup(textReaderArray[textReaderCurrent])
      _movie.sprite["textWindowInst"].title.text = titleTxt
      setRedirect("textReader")
    end if
  end if
end

on navTextReader(me, nav)
  if nav = "close" then
    me.hideTextReader()
  else if nav = "up" then
    _movie.sprite["textWindowInst"].textClip.scrollUp()
  else if nav = "down" then
    _movie.sprite["textWindowInst"].textClip.scrollDown()
  else if nav = "pageUp" then
    _movie.sprite["textWindowInst"].textClip.pageUp()
  else if nav = "pageDown" then
    _movie.sprite["textWindowInst"].textClip.pageDown()
  else if nav = "next" then
    if textReaderCurrent < count(textReaderArray) then
      textReaderCurrent = textReaderCurrent + 1
      _movie.sprite["textWindowInst"].textClip.textSetup(textReaderArray[textReaderCurrent])
      _movie.sprite["textWindowInst"].showLastButton("true")
      if textReaderCurrent = count(textReaderArray) then
        _movie.sprite["textWindowInst"].showNextButton("false")
      end if
    else if nav = "prev" then
      if textReaderCurrent > 1 then
        textReaderCurrent = textReaderCurrent - 1
        _movie.sprite["textWindowInst"].textClip.textSetup(textReaderArray[textReaderCurrent])
        _movie.sprite["textWindowInst"].showNextButton("true")
        if textReaderCurrent = 1 then _movie.sprite["textWindowInst"].showLastButton("false")
      end if
    end if
  end if
end

on hideTextReader(me)
  textReaderCurrent = 0
  _movie.sprite["textWindowInst"].visible = false
  _movie.sprite["textWindowInst"].ink = 0 --setting visibility to 0 (opaque) should help reduce memory
  setRedirect("")
end

--Fast text menu procedures

on initFTMenu(me)
  _movie.sprite["ftMenuInst"].visible = true

```

```

_movie.sprite["ftMenuInst"].ink = 100
end

on setFTmenu(me, colour, nameItems)
  _movie.sprite["ftMenuInst"].visible = true
  _movie.sprite["ftMenuInst"].ink = 100
  FTitems[colour] = nameItems
  --currentFTMenu = colour
  --currentFTMenuPos = 1
  --RGBAlpha settings for each colour
  colours = ["red":["200,0,0,100","255,0,0,100"],\
"green":["0,200,0,100","0,255,0,100"],\
"yellow":["255,200,0,100","255,230,0,100"],\
"blue":["0,200,255,100","0,255,255,100"]]
  nameItemsString = ""
  repeat with t in nameItems
    nameItemsString = nameItemsString & "," & quote & t & quote --converts array into string with commas and
quotes
  end repeat
  --srtring ver of flash command + colour arrays
  doPart1 = " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& colour &
"Menu.setupMenu(_movie.sprite["&quote&"ftMenuInst"&quote&"].newObject("&quote&"Array"&quote&","& colours[colour][1]
&"), _movie.sprite["&quote&"ftMenuInst"&quote&"].newObject("&quote&"Array"&quote&","& colours[colour][2]
&"), _movie.sprite["&quote&"ftMenuInst"&quote&"].newObject("&quote&"Array"&quote&","& colours[colour][3]
&"), _movie.sprite["&quote&"ftMenuInst"&quote&"].newObject("&quote&"Array"&quote&","& colours[colour][4]
&")"
  doPart2 = ""
  do doPart1 & nameItemsString & doPart2
  do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& colour & "Menu.showMe()"
end

on navFTMenu(me, coldir) --passed either a colour or direction
  colours = ["red","green","yellow","blue"]
  if coldir = "clearAll" then --hides menus and clears all - frames should run on frame exit
    _movie.sprite["ftMenuInst"].hideAll()
    FTitems = ["red":[],"green":[],"yellow":[],"blue":[]]
    currentFTMenuPos = 1
    currentFTMenu = ""
    setRedirect("") --tell ipmanager not to bypass keys
  else if coldir = "up" then --UP COMMAND
    if currentFTMenu <> "" then
      if currentFTMenuPos > 1 then
        currentFTMenuPos = currentFtMenuPos - 1
        do " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select("&
currentFTMenuPos &")"
      end if
    end if
  else if coldir = "down" then --DOWN COMMAND
    if currentFTMenu <> "" then
      if currentFTMenuPos < (count(FTitems[currentFTMenu]) - 1) then
        currentFTMenuPos = currentFtMenuPos + 1
        do " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select("&
currentFTMenuPos &")"
      end if
    end if
  else if coldir = "back" then --BACK COMMAND
    if currentFTMenu <> "" then
      do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select(1)"
      do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.close()"
      currentFTMenuPos = 1
      currentFTMenu = ""
      setRedirect("") --tell ipmanager not to bypass keys
    end if
  else if coldir = "enter" then --ENTER COMMAND
    if currentFTMenu <> "" then
      flashEvent((currentFTMenu&" "&FTitems[currentFTMenu][currentFTMenuPos + 1]))
      do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select(1)"
      do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.close()"
      currentFTMenuPos = 1
      currentFTMenu = ""
      setRedirect("") --tell ipmanager not to bypass keys
    end if
  else
    if coldir = currentFTMenu then --close current menu if same colour button pressed
      do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select(1)"
      do "_movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.close()"
      currentFTMenuPos = 1
      currentFTMenu = ""
      setRedirect("") --tell ipmanager not to bypass keys
    else
      if currentFTMenu <> "" then --if an existing menu is open then close it
        do " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select(1)"
        do " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.close()"
        currentFTMenuPos = 1
        currentFTMenu = ""
        setRedirect("") --tell ipmanager not to bypass keys
      end if
      if count(FTitems[coldir]) = 1 then --if there is only button text then exec and dont open menu
        flashEvent(coldir&"_"&FTitems[coldir][1])
      else --else open the menu and select item 1
        if FTitems[coldir] <> [] then --if menu not initialised then don't open it
          currentFTMenu = coldir
          currentFTMenuPos = 1
          do " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.open()"
          do " _movie.sprite["&quote&"ftMenuInst"&quote&"]."& currentFTMenu & "Menu.select(1)"
        end if
      end if
    end if
  end if
end

```

```

        setRedirect("FTMenu") --tell ipmanager to route keys here
    end if
end if
end if
end if
end

on hideFTMenu(me)
    _movie.sprite["ftMenuInst"].hideAll()
    _movie.sprite["ftMenuInst"].visible = false
    _movie.sprite["ftMenuInst"].ink = 0
    setRedirect("")
    FTItems = ["red": [], "green": [], "yellow": [], "blue": []]
end

```

CONFIG_MANAGER

```

--CONFIG_MANAGER
--this script loads the config file so other scripts can read the saved settings
--and overwrites the file with new configuration changes
--methods load, save, revert, plus accessing variables

global myFile

--properties can be accessed using config.property["propertyname"]
--a list of the information that needs to be saved
global cfgFile
property cfgList

global audioFile
property audioList

global imageFile
property videoList

on new(me)
    --this initialises variables that need to be set when an instance is made
    cfgFile = the moviepath & "settings.cfg"
    propSave = new(xtra "PropSave")
    --load
    return me
end

on load(me)
    cfgList = me.loadPropList(cfgFile)
    if integerP(cfgList) then me.loadDefaults()
    --when loading folder list(s) prompt if they can't be read
end

on save(me)
    me.savePropList(cfgList, cfgFile)
end

on loadDefaults(me)
    --all settings that require a default value should be listed here
    cfgList = ["folderRecursive":true,\
              "folderSort":true,\
              "dbUpdate":true]
end

-----
--scripts for saving property lists using propSave Xtra
--save requires a location and property list
--load requires a location and returns a property list
--load returns an error integer if prop list failed to load

on savePropList(me, prList, location)
    err = savePropsToDisk(location, prList)
    if integerP(err) then
        alert propSaveGetError(err)
    end if
end

on loadPropList(me, location)
    l = loadPropsFromDisk(location)
    if integerP(l) then
        alert propSaveGetError(l)
    end if
    return l
end

```

Behaviour Scripts

mainMenu

```

--mainmenu
global backHistory

global flMgr
global plAudio
global locations --locations of cd drive, memory card readers and favourite folders
property firstRun
global mainMenuStructure
global MMStructure

on isOKToAttach (me, aSpriteType, aSpriteNum)
    tIsOk = 0
    if aSpriteType = #script then
        tIsOk = 1
    end if
    return(tIsOk)
end on

--key commands

on keyUp()
    case (getKey()) of
        --add keys relevant to this frame
        "nowPlayingKey": gotoNowPlaying(the framelabel)
        "playKey": playAudio()
        "nextKey": plAudio.nextSong()
        "prevKey": plAudio.prevSong()
        "stopKey": stopAudio()
        "upKey": navUp()
        "downKey": navDown()
        "leftKey": navBack()
        "rightKey": navSelect()
        "enterKey": navSelect()
        "backKey": navBack()
        "redKey": flMgr.navFTMenu("red")
        "greenKey": flMgr.navFTMenu("green")
        "yellowKey": flMgr.navFTMenu("yellow")
        "blueKey": flMgr.navFTMenu("blue")
    end case
end

on mouseUp()
    click = getMouseClick()
    case (click[1]) of
        --add flash buttons relevant to frame
        --"menu0": menu0(click[2])
    end case
end

on beginSprite(flaMMInst)
    firstRun = true
    --used to run code when a frame is first entered
end

on endSprite(flaMMInst)
    --run once on exit
    flMgr.hideFTMenu()
end

on exitFrame(me)
    go the frame
    flEvent = getFlashEvent()
    audioTicker() --keeps song changing/playlist running
    if flEvent <> "" then
        case (flEvent) of
            "red_Power Off": flMgr.showMsgBox("Quit", "Are you sure you want to quit the media
centre?", "question", ["Yes", "No"], 2)
            "msgBox_Quit_Yes": quit
        end case
    end if
end

on enterFrame()
    if firstRun = true then
        firstRun = false
        member("title").text = "Media Guide"
        initMainMenu()
        mainMenuStructure = MMStructure
        --set item names in menu
        menuBackRGB = _movie.sprite["flaMMInst"].newObject("Array", 210, 230, 255, 50)
        globalEdgeRGB = _movie.sprite["flaMMInst"].newObject("Array", 210, 230, 255, 100)
        menu0colour = _movie.sprite["flaMMInst"].newObject("Array", 0, 230, 230, 100)
        menu1colour = _movie.sprite["flaMMInst"].newObject("Array", 0, 230, 0, 100)
        menu2colour = _movie.sprite["flaMMInst"].newObject("Array", 230, 230, 0, 100)
        menu3colour = _movie.sprite["flaMMInst"].newObject("Array", 230, 0, 230, 100)
        menu4colour = _movie.sprite["flaMMInst"].newObject("Array", 255, 200, 0, 100)
        menu5colour = _movie.sprite["flaMMInst"].newObject("Array", 0, 0, 255, 100)

        mainMenuItems = _movie.sprite["flaMMInst"].newObject("Array", \
        _movie.sprite["flaMMInst"].newObject("Array", mainMenuStructure.getPropAt(1), globalEdgeRGB, menu0colour), \
        _movie.sprite["flaMMInst"].newObject("Array", mainMenuStructure.getPropAt(2), globalEdgeRGB, menu1colour), \
        movie.sprite["flaMMInst"].newObject("Array", mainMenuStructure.getPropAt(3), globalEdgeRGB, menu2colour), \

```

```

_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure.getPropAt(4),globalEdgeRGB,menu3colour),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure.getPropAt(5),globalEdgeRGB,menu4colour),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure.getPropAt(6),globalEdgeRGB,menu5colour))
    button0Items = _movie.sprite["flaMMInst"].newObject("Array",\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[1][1].getPropAt(1),globalEdgeRGB,menu0colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[1][1].getPropAt(2),globalEdgeRGB,menu0colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[1][1].getPropAt(3),globalEdgeRGB,menu0colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[1][1].getPropAt(4),globalEdgeRGB,menu0colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[1][1].getPropAt(5),globalEdgeRGB,menu0colou
r))
    button1Items = _movie.sprite["flaMMInst"].newObject("Array",\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[2][1].getPropAt(1),globalEdgeRGB,menu1colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[2][1].getPropAt(2),globalEdgeRGB,menu1colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[2][1].getPropAt(3),globalEdgeRGB,menu1colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[2][1].getPropAt(4),globalEdgeRGB,menu1colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[2][1].getPropAt(5),globalEdgeRGB,menu1colou
r))
    button2Items = _movie.sprite["flaMMInst"].newObject("Array",\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[3][1].getPropAt(1),globalEdgeRGB,menu2colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[3][1].getPropAt(2),globalEdgeRGB,menu2colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[3][1].getPropAt(3),globalEdgeRGB,menu2colou
r))
    button3Items = _movie.sprite["flaMMInst"].newObject("Array",\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[4][1].getPropAt(1),globalEdgeRGB,menu3colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[4][1].getPropAt(2),globalEdgeRGB,menu3colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[4][1].getPropAt(3),globalEdgeRGB,menu3colou
r))
    button4Items = _movie.sprite["flaMMInst"].newObject("Array",\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[5][1].getPropAt(1),globalEdgeRGB,menu4colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[5][1].getPropAt(2),globalEdgeRGB,menu4colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[5][1].getPropAt(3),globalEdgeRGB,menu4colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[5][1].getPropAt(4),globalEdgeRGB,menu4colou
r))
    button5Items = _movie.sprite["flaMMInst"].newObject("Array",\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[6][1].getPropAt(1),globalEdgeRGB,menu5colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[6][1].getPropAt(2),globalEdgeRGB,menu5colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[6][1].getPropAt(3),globalEdgeRGB,menu5colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[6][1].getPropAt(4),globalEdgeRGB,menu5colou
r),\
_movie.sprite["flaMMInst"].newObject("Array",mainMenuStructure[6][1].getPropAt(5),globalEdgeRGB,menu5colou
r))
    --group the menu together
    mnuStructure = _movie.sprite["flaMMInst"].newObject("Array",\
    mainMenuItems,\
    button0Items,\
    button1Items,\
    button2Items,\
    button3Items,\
    button4Items,\
    button5Items)

```

```

--init the menu system
_movie.sprite["flaMMInst"].initMenuSystem(menuBackRGB,menuBackRGB,mnuStructure)
--display main menu
_movie.sprite["flaMMInst"].viewMenu(0)
_movie.sprite["flaMMInst"].navMenu(0,0) --set flash menu to 0
--FASTTEXT MENU
flMgr.setFTMenu("red",["Power Off"])
--CLOCK COLOUR INIT
_movie.sprite["clockInst"].setColour(_movie.sprite["clockInst"].newObject("Array",210, 230, 255, 100))
end if
end

on navUp()
repeat with n = 1 to count(mainMenuStructure)
if mainMenuStructure[n][2] = 1 then
if n = 1 then
exit repeat --if at top of menu do nothing
else
mainMenuStructure[n][2] = 0
mainMenuStructure[n-1][2] = 1 --select item above
_movie.sprite["flaMMInst"].navMenu(0,(n-1)-1) --extra -1 as flash works with 0's
exit repeat
end if
else
repeat with s = 1 to count(mainMenuStructure[n][1])
if mainMenuStructure[n][1][s][2] = 1 then
if s = 1 then
exit repeat --if at top of submenu do nothing
else
mainMenuStructure[n][1][s][2] = 0
mainMenuStructure[n][1][s-1][2] = 1 --select item above
_movie.sprite["flaMMInst"].navMenu(n,(s-1)-1)
exit repeat
end if
end if
end repeat
end if
end repeat
end

on navDown()
repeat with n = 1 to count(mainMenuStructure)
if mainMenuStructure[n][2] = 1 then
if n = count(mainMenuStructure) then
exit repeat --if at bottom of menu do nothing
else
mainMenuStructure[n][2] = 0
mainMenuStructure[n+1][2] = 1 --select item below
_movie.sprite["flaMMInst"].navMenu(0,(n+1)-1) --extra -1 for flash
exit repeat
end if
else
repeat with s = 1 to count(mainMenuStructure[n][1])
if mainMenuStructure[n][1][s][2] = 1 then
if s = count(mainMenuStructure[n][1]) then
exit repeat --if at bottom of submenu do nothing
else
mainMenuStructure[n][1][s][2] = 0
mainMenuStructure[n][1][s+1][2] = 1 --select item below
_movie.sprite["flaMMInst"].navMenu(n,(s+1)-1)
exit repeat
end if
end if
end repeat
end if
end repeat
end

on navSelect()
mainMenuSelect = 0
subMenuSelect = 0
repeat with n = 1 to count(mainMenuStructure)
if mainMenuStructure[n][2] = 1 then
mainMenuSelect = n
mainMenuStructure[n][2] = 0 --no longer in the main menu - set to 0
mainMenuStructure[n][1][1][2] = 1 --always go to first item in submenu
_movie.sprite["flaMMInst"].navMenu(n,0)
_movie.sprite["flaMMInst"].viewMenu(n)
exit repeat
else
repeat with s = 1 to count(mainMenuStructure[n][1])
if mainMenuStructure[n][1][s][2] = 1 then
subMenuSelect = mainMenuStructure[n][1][s][1]
exit repeat
end if
end repeat
end if
end repeat
if subMenuSelect <> 0 then
--actions for the menu items
case (subMenuSelect) of
"viewcdall": browse(locations["cdRom"],"all",the frameLabel)
"viewcdmusic": browse(locations["cdRom"],"music",the frameLabel)
"viewcdvideos": browse(locations["cdRom"],"video",the frameLabel)
"viewcdpictures": browse(locations["cdRom"],"image",the frameLabel)
"artists": browseLibrary("artist",the frameLabel)

```

```

"albums": browseLibrary("album",the frameLabel)
"genres": browseLibrary("genre",the frameLabel)
"songs": browseLibrary("song",the frameLabel)
"films":browse(locations["filmFolder"],"video",the frameLabel)
"episode":browse(locations["episodeFolder"],"video",the frameLabel)
"musicvideo":browse(locations["musicVideoFolder"],"video",the frameLabel)
"viewmcall": browse(locations["memoryCard"],"all",the frameLabel)
"viewmcmusic": browse(locations["memoryCard"],"music",the frameLabel)
"viewmcvideos": browse(locations["memoryCard"],"video",the frameLabel)
"viewmcpictures": browse(locations["memoryCard"],"image",the frameLabel)
"browsepicture": browsePicture(locations["pictureFolder"],the framelabel)
end case
end if
end

on navBack()
repeat with n = 1 to count(mainMenuStructure)
if mainMenuStructure[n][2] = 1 then
--put "already at main menu cannot go back"
if backHistory[the frameLabel] <> "" then _movie.go(_movie.label(backHistory[the frameLabel]))
exit repeat
else
repeat with s = 1 to count(mainMenuStructure[n][1])
if mainMenuStructure[n][1][s][2] = 1 then
mainMenuStructure[n][1][s][2] = 0
mainMenuStructure[n][2] = 1 --go back to main menu (of this submenu item)
_movie.sprite["flaMMInst"].navMenu(0,n-1)
_movie.sprite["flaMMInst"].viewMenu(0)
put "gone back to menu level " && n
exit repeat
end if
end repeat
end if
end repeat
end

```

library

```

--library
global backHistory
global flMgr
global dbMgr
global firstRun
global plAudio
global fileMgr

global searchMode --specifies if the predictive window is open
global searchText --holds the numbers entered into the predictiv window
global searchResults --holds the matches returned by the predictive window
global searchWord --stores the highlighted word
global searchWordNo --stores the highlighted word number
global searchCharsTable -- basic table similar to one used in predictive db to put letters in place of
numbers

property itemsOnPage

global currentNavType
global leftPanel
global rightPanel
global currentPanel

--global selectAll --0=not on first page, 1=yes, display 'all', 2=yes and select all highlighted

global panelTitles
global panelCurrentSelOnPage --calculated at display-time but saved so play/enqueue knows which item is
selected
global panelCurrentSel --array - stores highlighted item no of each panel
global panelCurrentItems --array to keep the items so the id's etc can be got whenever
global panelTotalItems --total number of items in each panel
global panelSelectAll --if 'all' is chosen from top of panel

on isOKToAttach (me, aSpriteType, aSpriteNum)
tIsOk = 0
if aSpriteType = #script then
tIsOK = 1
end if
return(tIsOK)
end on

--key commands
on keyUp()
case (getKey()) of
--add keys relevant to this frame
"nowPlayingKey": gotoNowPlaying(the framelabel)
"powerKey": gotoMenu(the framelabel)
"playKey":playAudio()
"nextKey":plAudio.nextSong()
"prevKey":plAudio.prevSong()
"stopKey":stopAudio()
"1Key": keypad("1")
"2Key": keypad("2")
"3Key": keypad("3")
"4Key": keypad("4")

```

```

"5Key": keypad("5")
"6Key": keypad("6")
"7Key": keypad("7")
"8Key": keypad("8")
"9Key": keypad("9")
"0Key": keypad("0")
"delKey": keypad("del")
"wordKey": keypad("word")
"infoKey": navShowInfo()
"upKey": navUp()
"downKey": navDown()
"leftKey": navBack()
"rightKey": navSelect()
"enterKey": navSelect()
"pageUpKey": navPageUp()
"pageDownKey": navPageDown()
"backKey": navBack()
"redKey": flMgr.navFTMenu("red")
"greenKey": flMgr.navFTMenu("green")
"yellowKey": flMgr.navFTMenu("yellow")
"blueKey": flMgr.navFTMenu("blue")
end case
end

on mouseUp()
click = getMouseClicked()
case (click[1]) of
--add flash buttons relevant to frame
"menu0": menu0(click[2])
end case
end

on beginSprite(libraryInst)
firstRun = true
end

on endSprite(libraryInst)
flMgr.hideFTMenu()
_movie.sprite["prdWindowInst"].visible = false
end

on exitFrame(me)
flEvent = getFlashEvent()
if flEvent <> "" then
case (flEvent) of
--message boxes
"msgBox_Clear Music Playlist_Yes": plAudio.clearPlaylist()
"msgBox_Play Music In Folder_Yes": --playFolder(true, folderArray)
"msgBox_Play Music File_Yes": -- playTheFile()
"msgBox_Play Selection_Yes": navQueue(true) --dont queue but play
--view by file type
"red_By artist": setNavigation("artist")
"red_By album": setNavigation("album")
"red_By genre": setNavigation("genre")
"red_By song": setNavigation("song")
--playlist control
"green_View": backHistory["sound"] = the frameLabel
_movie.go(_movie.label("sound"))
"green_Clear": flMgr.showMsgBox("Clear Music Playlist", "Are you sure you want to clear the current
music playlist?", "question", ["Yes", "No"], 2)
--folder all files
"yellow_Play Item": navPlayCheck()
"blue_Queue Item": navQueue(false) --queue
end case
end if
go the frame
end

on enterFrame()
if firstRun = true then
firstRun = false
searchCharsTable = [
"1": "?", \
"2": "a", \
"3": "d", \
"4": "g", \
"5": "j", \
"6": "m", \
"7": "p", \
"8": "t", \
"9": "w", \
"0": " "]
searchText = ""
itemsOnPage = 12
leftPanel = ""
rightPanel = ""
currentPanel = ""
panelTitles = ["0": "", "1": "", "2": "", "3": ""]
panelCurrentSel = ["0": 0, "1": 0, "2": 0, "3": 0]
panelSelectAll = ["0": 0, "1": 0, "2": 0, "3": 0]
panelCurrentSelOnPage = ["0": 0, "1": 0, "2": 0, "3": 0]
panelCurrentItems = ["0": [], "1": [], "2": [], "3": []]
panelTotalItems = ["0": 0, "1": 0, "2": 0, "3": 0]
_movie.sprite["libraryInst"].setColours(0, 75, 255, 100, 75, 255, 255, 10)
_movie.sprite["libraryInst"].initPanels(30)
setNavigation(currentNavType)

```

```

    updateFTMenu()
    hidePrdWindow()
end if
end

on setNavigation(navType) --reset navigation, can be genre,artist,album,song
    _movie.sprite["libraryInst"].hideAll() --hide all existing panels
    --set the navigation type
    if navType = "genre" then
        currentNavType =
["0":["Genre",1,"GenreID","GenreName","L"],"1":["Artist",0,"ArtistID","ArtistName",""],"2":["Album",0,"AlbumID","AlbumName",""],"3":["Song",0,"SongID","Title",""]]
    else if navType = "artist" then
        currentNavType =
["0":["Artist",1,"ArtistID","ArtistName","L"],"1":["Album",0,"AlbumID","AlbumName",""],"2":["Song",0,"SongID","Title",""]]
    else if navType = "album" then
        currentNavType = ["0":["Album",1,"AlbumID","AlbumName","L"],"1":["Song",0,"SongID","Title",""]]
    else if navType = "song" then
        currentNavType = ["0":["Song",1,"SongID","Title","L"]]
    end if
    currentPanel = "0"
    panelCurrentSel[currentPanel] = 1
    panelTotalItems[currentPanel] =
dbMgr.setPanel(currentPanel,currentNavType[currentPanel][1],currentNavType[currentPanel][3],currentNavType
[currentPanel][4],"",0)
    --leftPanel = "0"
    --rightPanel = ""
    --set titles of all panels
    navCount = 0
    repeat with i=1 to count(currentNavType)
        panelTitles[string(navCount)] = currentNavType[i][1]
        navCount = navCount + 1
    end repeat
    _movie.sprite["libraryInst"].slideToPanel(0)
    panelDisplay(currentPanel,true)
    hidePrdWindow()
end

on navPlayCheck()
    plSize = count(plAudio.getPl())
    if plSize > 1 then
        flMgr.showMsgBox("Play Selection","The current music playlist contains " & plSize & " songs. Are you
sure you want to play the folder?","question",["Yes","No"],2)
    else
        --only 1 file, replace anyway
        navQueue(true)
    end if
end

on navQueue(playMode) --false for queue, true for play
    songPath = ""
    if panelSelectAll[currentPanel] = 2 then
        sel = dbMgr.getPanelPage(currentPanel,1,panelTotalItems[currentPanel])
    else
        dbResult = dbMgr.getSong(panelCurrentItems[currentPanel][(panelCurrentSelOnPage[currentPanel])][2])
        songPath = dbResult[1][2]&dbResult[1][3]
    end if
    if playMode = true then plAudio.clearPlaylist()
    queueUp(true,0,songPath)
    if playMode = true then plAudio.advance()
end

on navShowInfo()
    dbResult = dbMgr.getSong(panelCurrentItems[currentPanel][(panelCurrentSelOnPage[currentPanel])][2])
    songPath = dbResult[1][2]&dbResult[1][3]
    titleTxt = "Info for " & songPath
    bodyTxt = fileMgr.getInfo(songPath,false)
    flMgr.showTextReader(titleTxt,[bodyTxt])
end

on navUp()
    if panelSelectAll[currentPanel] <> 2 then --if 'all' highlighted (selectAll = 2) then cant move up
        if panelSelectAll[currentPanel] = 1 and panelCurrentSel[currentPanel] = 1 then
            --if just below 'all' then move up to 'all'
            panelSelectAll[currentPanel] = 2
            panelDisplay(currentPanel,false)
        else
            --carry on as normal
            if panelCurrentSel[currentPanel] > 1 then
                panelCurrentSel[currentPanel] = panelCurrentSel[currentPanel] - 1
                panelDisplay(currentPanel,false)
            end if
        end if
    end if
end

on navDown()
    if panelSelectAll[currentPanel] = 2 then
        panelSelectAll[currentPanel] = 1
        panelDisplay(currentPanel,false)
    else
        if panelCurrentSel[currentPanel] < panelTotalItems[currentPanel] then
            panelCurrentSel[currentPanel] = panelCurrentSel[currentPanel] + 1
        end if
    end if
end

```

```

        panelDisplay(currentPanel, false)
    end if
end if
end

on navPageUp()
    if panelCurrentSel[currentPanel] > 1 then
        if (panelCurrentSel[currentPanel] - itemsOnPage) <= 1 then
            --less than a page from the top, go to top
            panelCurrentSel[currentPanel] = 1
            panelDisplay(currentPanel, false)
        else
            --move n items up
            panelCurrentSel[currentPanel] = panelCurrentSel[currentPanel] - itemsOnPage
            panelDisplay(currentPanel, false)
        end if
    end if
end

on navPageDown()
    if panelCurrentSel[currentPanel] < panelTotalItems[currentPanel] then
        if (panelCurrentSel[currentPanel] + itemsOnPage) >= panelTotalItems[currentPanel] then
            --less than a page from the bottomw, go to bottom
            panelCurrentSel[currentPanel] = panelTotalItems[currentPanel]
            panelDisplay(currentPanel, false)
        else
            --move n items down
            panelCurrentSel[currentPanel] = panelCurrentSel[currentPanel] + itemsOnPage
            panelDisplay(currentPanel, false)
        end if
    end if
end

--used to go to a record
on navGoto(no)
    if not voidP(no) then
        if no <= panelTotalItems[currentPanel] then
            if panelSelectAll[currentPanel] = 2 then
                panelSelectAll[currentPanel] = 1
            end if
            panelCurrentSel[currentPanel] = no
            panelDisplay(currentPanel, false)
        end if
    end if
end

on navSelect() --equivalent to moving right
    hidePrdWindow()
    repeat with i=1 to count(currentNavType)
        if currentNavType[i][2] = 1 then
            if i = count(currentNavType) then
                --cannot browse right activate the item instead
                exit repeat
            else
                --update focus of the new panel/set current panel variable
                newPanel = currentNavType.getPropAt(i+1)
                currentNavType[i][2] = 0
                currentNavType[i+1][2] = 1
                --browse right to the next panel
                if currentNavType[i][5] = "L" then
                    --panel on left hand side - no need to move
                else
                    --panel on right hand side
                    currentNavType[i-1][5] = "" --old left hand panel is now off-screen
                    currentNavType[i][5] = "L" --update array to say its now on the left
                    _movie.sprite["libraryInst"].slideToPanel(integer(currentPanel)) --move it to the left
                end if
                currentNavType[i+1][5] = "R" --new panel will always open on the right
                panelCurrentSel[newPanel] = 1
                if panelSelectAll[currentPanel] = 2 then
                    refined = false
                    --don't refine by the last panel, but check for previous refines
                    if (i - 1) > 0 then
                        if panelSelectAll[(i - 1)] <> 2 then
                            --DOSOMTHIN
                            refineID = panelCurrentItems[(i - 1)][(panelCurrentSelOnPage[(i - 1))]] [2]
                            refineField = currentNavType[i-1][3]
                            panelTotalItems[newPanel] =
dbMgr.setPanel(newPanel, currentNavType[newPanel][1], currentNavType[newPanel][3], currentNavType[newPanel][4]
], refineField, refineID)
                            refined = true
                        else
                            if (i - 2) > 0 then
                                if panelSelectAll[(i - 2)] <> 2 then
                                    refineID = panelCurrentItems[(i - 2)][(panelCurrentSelOnPage[(i - 2))]] [2]
                                    refineField = currentNavType[i-2][3]
                                    panelTotalItems[newPanel] =
dbMgr.setPanel(newPanel, currentNavType[newPanel][1], currentNavType[newPanel][3], currentNavType[newPanel][4]
], refineField, refineID)
                                    refined = true
                                end if
                            else
                                end if
                            end if
                        end if
                    end if
                end if
            end if
        end if
    end if
end if

```

```

        end if
        if refined = false then
            --don't refine by anything
            panelTotalItems[newPanel] =
dbMgr.setPanel(newPanel, currentNavType[newPanel][1], currentNavType[newPanel][3], currentNavType[newPanel][4
], "", 0)
        end if
    else
        --refine by last panel
        refineID = panelCurrentItems[currentPanel][(panelCurrentSelOnPage[currentPanel])][2]
        refineField = currentNavType[i][3]
        panelTotalItems[newPanel] =
dbMgr.setPanel(newPanel, currentNavType[newPanel][1], currentNavType[newPanel][3], currentNavType[newPanel][4
], refineField, refineID)
    end if
    --update current pannel var and draw new panel
    currentPanel = newPanel
    panelDisplay(currentPanel, true)
    exit repeat
end if
end if
end repeat
end

on navBack()
    if searchMode = true then
        hidePrdWindow()
    else
        repeat with i = 1 to count(currentNavType)
            if currentNavType[i][2] = 1 then
                if i = 1 then
                    --cannot go up
                    exit repeat
                else
                    newPanel = currentNavType.getPropAt(i-1)
                    currentNavType[i][2] = 0
                    currentNavType[i-1][2] = 1
                    if currentNavType[i][5] = "R" then
                        --panel on right hand side - no need to move
                    else
                        --move panel to right
                        currentNavType[i+1][5] = ""
                        currentNavType[i][5] = "R"
                    end if
                    currentNavType[i-1][5] = "L"
                    _movie.sprite["libraryInst"].hidePanel(integer(currentPanel))
                    currentPanel = newPanel
                    _movie.sprite["libraryInst"].slideToPanel(integer(currentPanel))
                    panelDisplay(currentPanel, false)
                end if
            end if
        end repeat
    end if
end

on hidePrdWindow()
    searchMode = false
    searchText = ""
    searchWord = ""
    searchWordNo = 1
    _movie.sprite["prdWindowInst"].visible = false
end

on keypad(keypress)
    --manages predictive serarch query/results
    --search field is managed in string then transferred to flash
    if searchMode = false and keyPress = "word" and keypress = "del" then
        --do nothing
    else
        if searchMode = false then
            --the window was hidden
            --position it
            _movie.sprite["prdWindowInst"].setColours(0, 75, 255, 100, 0, 100, 200, 90)
            member("predictiveWindow").regPoint = point(0,0)
            if currentNavType[currentPanel][5] = "R" then _movie.sprite["prdWindowInst"].locH = 217
            if currentNavType[currentPanel][5] = "L" then _movie.sprite["prdWindowInst"].locH = 378
            --show it
            searchMode = true
            _movie.sprite["prdWindowInst"].visible = true
            _movie.sprite["prdWindowInst"].title.text.text = "Searching " & panelTitles[currentPanel]
            searchText = ""
            searchWordNo = 1
            searchWord = ""
        end if
        if keypress = "del" then
            searchText = chars(searchText, 1, searchText.length - 1) -- delete 1 char
            if searchText = "" then
                hidePrdWindow()
                exit
            end if
        else if keypress = "word" then
            if searchWordNo + 1 <= count(searchResults) then
                searchWordNo = searchWordNo + 1
                if searchWordNo = 7 then searchWordNo = 1
                searchWord = searchResults[searchWordNo][2]
            end if
        end if
    end
end

```

```

end if
else
  searchText = searchText & keypress
end if
searchResults = dbMgr.predictiveSearch(currentPanel, searchText)
if not voidP(searchResults) then
  if count(searchResults) > 0 then
    --see if the last highlighted word is still in the list
    if searchWord <> "" then
      found = false
      r = 6
      if count(searchResults) < 6 then r = count(searchResults)
      repeat with i = 1 to r
        if searchResults[i][2] = searchWord then
          searchWordNo = i
          found = true
        end if
      end repeat
      if found = false then
        searchWord = ""
        searchWordNo = 1
      end if
    end if
    --scroll to highlighted result in list
    rowNo = dbMgr.getPanelRowNo(currentPanel, searchResults[searchWordNo][2])
    navGoto(rowNo)
  end if
end if
--create text for display
searchDispText = ""
if searchText.length > 0 then
  if count(searchResults) > 0 then
    searchDispText = chars(searchResults[searchWordNo][2], 1, searchText.length)
  else
    repeat with i=1 to searchText.length
      searchDispText = searchDispText & searchCharsTable[chars(searchText, i, i)]
    end repeat
    searchDispText = searchDispText & " (no matches)"
    _movie.sprite["prdWindowInst"].setHighlight(10)
  end if
end if
setPrdWindow(searchResults, searchWordNo, searchDispText)
end if
end

--accepts array type [[no,name],[...,...]]
on setPrdWindow(suggestions, sel, inputText)
  if not voidP(suggestions) then
    if not voidP(sel) then
      c = 6
      r = 0
      if count(suggestions) < 6 then
        c = count(suggestions)
        r = 6 - count(suggestions)
      end if
      arrayStr = ""
      repeat with i = 1 to c
        if i > 1 then arrayStr = arrayStr & ","
        arrayStr = arrayStr && QUOTE&suggestions[i][2]&QUOTE
      end repeat
      if r > 0 then
        repeat with i = 1 to r
          arrayStr = arrayStr & "," && QUOTE&QUOTE
        end repeat
      end if
      _movie.sprite["prdWindowInst"].inputText.text = inputText
      do " _movie.sprite["&QUOTE&"prdWindowInst"&QUOTE&"].setSuggestions("&arrayStr&")
      _movie.sprite["prdWindowInst"].setHighlight(sel - 1)
    end if
  end if
end

on updateFTMenu()
  flMgr.hideFTMenu()
  flMgr.setFTMenu("red", ["Change view", "By artist", "By album", "By genre", "By song"])
  flMgr.setFTMenu("green", ["Playlist", "View", "Clear"])
  flMgr.setFTMenu("yellow", ["Play Item"])
  flMgr.setFTMenu("blue", ["Queue Item"])
end

on panelDisplay(panelNo, firstInit)
  noItems = "No items"
  noPages = "No pages"
  pages = ceil(panelTotalItems[panelNo] / (itemsOnPage + 0.0))
  page = 1
  selOnPage = 0
  if panelTotalItems[panelNo] > 0 then
    if firstInit = true then
      --first time panel is initialised
      --will be on page 1 and 'select all' is displayed
      panelSelectAll[currentPanel] = 2
    end if
    page = ceil((panelCurrentSel[panelNo] / (itemsOnPage + 0.0)))
    noPages = "Page " & page & " of " & pages
    noItems = panelCurrentSel[panelNo] & " / " & panelTotalItems[panelNo]
  end if
end

```

```

selOnPage = panelCurrentSel[panelNo] - (itemsOnPage*(page-1))
panelCurrentSelOnPage[panelNo] = selOnPage
pageItems = []
if (page * itemsOnPage) > panelTotalItems[panelNo] then
  --page will not be full
  panelCurrentItems[panelNo] = dbMgr.getPanelPage(panelNo,((page * itemsOnPage) - (itemsOnPage -
1)),panelTotalItems[panelNo])
else
  --page will be full
  panelCurrentItems[panelNo] = dbMgr.getPanelPage(panelNo,((page * itemsOnPage) - (itemsOnPage -
1)),(page * itemsOnPage))
end if
tempArray = []
repeat with a = 1 to count(panelCurrentItems[panelNo])
  tempArray.add(panelCurrentItems[panelNo][a][3])
end repeat
if page > 1 then panelSelectAll[currentPanel] = 0
if page = 1 and panelSelectAll[currentPanel] = 0 then panelSelectAll[currentPanel] = 1
itemsString = ""
if panelSelectAll[currentPanel] > 0 then itemsString = itemsString & ", " &QUOTE& "(All " &
panelTitles[currentPanel]&s)"&QUOTE
itemsString = itemsString & arrayToString(tempArray)
do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].setupPanel"&panelNo&"("&QUOTE&
panelTitles[panelNo]&QUOTE& ", "&QUOTE&QUOTE& ", "&QUOTE& noItems&QUOTE& ", "&QUOTE& noPages&QUOTE&
", _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE& itemsString & ")")
--first page shows 'All'
if panelSelectAll[currentPanel] = 2 then
  --'all' highlighted - first item in list
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].panel"&panelNo&".select(0)"
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].panel"&panelNo&".selectArrow(0)"
else if panelSelectAll[currentPanel] = 1 then
  --'all' not highlighted - highlight item + 1
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].panel"&panelNo&".select("& (selOnPage) &")"
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].panel"&panelNo&".selectArrow("& (selOnPage) &")"
else
  --'all' not on page - carry on as normal
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].panel"&panelNo&".select("& (selOnPage - 1) &")"
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].panel"&panelNo&".selectArrow("& (selOnPage - 1)
&")"
end if
else
  --do stuff when the page is empty
  do " _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].setupPanel"&panelNo&"("&QUOTE&
panelTitles[panelNo]&QUOTE& ", "&QUOTE&QUOTE& ", "&QUOTE& noItems&QUOTE& ", "&QUOTE& noPages&QUOTE&
", _movie.sprite["&QUOTE&"libraryInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE& ")")
end if
end

on arrayToString(theArray)
  --converts array into string(used for sending Flash Arrays)
  if not voidP(theArray) then
    theString = ""
    repeat with i=1 to count(theArray)
      theString = theString & ", " &QUOTE& searchAndReplace(theArray[i],QUOTE,"")&QUOTE
    end repeat
    return theString
  end if
end

on searchAndReplace(input, oldStr, newStr)
  -- searches the input (string) for oldStr and replaces it with newStr
  set output = ""
  repeat while input contains oldStr
    set posn = offset(oldStr, input)-1
    if posn > 0 then put char 1 to posn of input after output
    put newStr after output
    delete char 1 to (posn + length(oldStr)) of input
  end repeat
  put input after output
  return output
end

```

sound

```

--sound
global backHistory

global flMgr
global plAudio
global fileMgr
global repeatMode

global artMode --boolean - if art or vis is shown

global firstRun
global editMode
global elapsedAudioTime
global currentAudioPlayMode
global lastAudioPlayMode --used to prevent unnessary flash updating unless play mode has changed
global currentAudioFileTags
global currentAudioFile

```

```

global tempPlaylist -- restored if the modifications are discarded

on isOKToAttach (me, aSpriteType, aSpriteNum)
  tIsOk = 0
  if aSpriteType = #script then
    tIsOk = 1
  end if
  return(tIsOk)
end on

--key commands
on keyUp()
  case (getKey()) of
    --add keys relevant to this frame
    "powerKey": gotoMenu(the framelabel)
    "infoKey": navShowInfo()
    "playKey": playAudio()
    "nextKey": if plAudio.nextSong() = true then refreshItems()
    "prevKey": if plAudio.prevSong() = true then refreshItems()
    "stopKey": stopAudio()
    "upKey": navUp()
    "downKey": navDown()
    "leftKey": rewAudio()
    "rightKey": ffAudio()
    "enterKey": navEnter()
    "backKey": navBackKey()
    "redKey": flMgr.navFTMenu("red")
    "greenKey": flMgr.navFTMenu("green")
    "yellowKey": flMgr.navFTMenu("yellow")
    "blueKey": flMgr.navFTMenu("blue")
    "pageUpKey": navPageUp()
    "pageDownKey": navPageDown()
  end case
end

on mouseUp()
  click = getMouseClicked()
  case (click[1]) of
    --add flash buttons relevant to frame
    "menu0":
  end case
end

on beginSprite(playlistInst)
  firstRun = true
end

on endSprite(playlistInst)
  flMgr.hideFTMenu()
end

on exitFrame(me)
  go the frame
  if audioTicker() = true then refreshItems()
  evt = msecToHMS(getCurrentTime())
  --elapsed time update
  if elapsedAudioTime <> evt then
    if currentAudioPlayMode = 2 then --information bogus if not playing back
      elapsedAudioTime = evt
      _movie.sprite["audioOSDInst"].time.text.text = evt
      _movie.sprite["audioOSDInst"].progressBar.setProgress((playPercent(getCurrentTime()),getDuration()))
    end if
  end if
  --icon update
  if currentAudioPlayMode <> lastAudioPlayMode then
    lastAudioPlayMode = currentAudioPlayMode
    if currentAudioPlayMode = 2 then
      _movie.sprite["audioPlayIconInst"].playbackIcon.setIcon("play")
    else if currentAudioPlayMode = 3 then
      _movie.sprite["audioPlayIconInst"].playbackIcon.setIcon("pause")
    else
      --stop/nofile mode
      _movie.sprite["audioPlayIconInst"].playbackIcon.setIcon("stop")
    end if
  end if
  flEvent = getFlashEvent()
  if flEvent <> "" then
    case (flEvent) of
      --view by file type
      --edit mode
      "red_Remove": navDelete()
      "green_Select None": navHighlightNone()
      "yellow_Move Down": navMoveDown()
      "blue_Move Up": navMoveUp()
      "blue_Move Down": navMoveUp()
      --normal mode
      "red_Edit Playlist": setEditMode()
      "green_Save":
      "green_Load":
      "green_Clear": flMgr.showMsgBox("Clear Playlist","Are you sure you want to clear the current
playlist?","question",[ "Yes","No"],2)
      "msgBox_Clear Playlist Yes": plAudio.clearPlaylist()
      "yellow_Randomise": randomise()
      "yellow_Alphabetical": sortAlphabetical()
      "blue Repeat Off": repeatMode = 0
    end case
  end if
end

```

```

    "blue_Repeat All": repeatMode = 1
    "blue_Repeat Track": RepeatMode = 2
    "msgBox_Save Changes_Yes": exitEditMode(true) --boolean = save changes
    "msgBox_Save Changes_No": exitEditMode(false)
    "msgBox_Clear Playlist_Yes": if plAudio.clearPlaylist() = true then refreshItems()
end case
end if
end

on enterFrame()
if firstRun = true then
firstRun = false
editMode = false
tempPlaylist = []
_movie.sprite["playlistInst"].setColours(0,75,255,100,75,255,255,10)
_movie.sprite["audioOSDInst"].setColours(0,75,255,100,75,255,255,10)
_movie.sprite["audioPlayIconInst"].setColours(0,75,255,100,75,255,255,10)
_movie.sprite["audioPlayIconInst"].playbackIcon.setIcon("stop")
member("title").text = "Now Playing"
refreshItems()
updateFTMenu()
end if
end

on setEditMode()
showEditItems(true)
editMode = true
tempPlaylist = plAudio.getPl().duplicate()
updateFTMenu()
updatePLWindow()
--do other stuff like show the "press back to exit"
end

on exitEditMode(saveChanges)
showEditItems(false)
editMode = false
updateFTMenu()
if saveChanges = false then
plAudio.setPl(tempPlaylist.duplicate())
end if
plAudio.highlightNone()
updatePLWindow()
end

on navShowInfo()
currentSelection = plAudio.navGetSelected()
if not voidP(currentSelection) then
titleTxt = "Info for " & currentSelection[1]
bodyTxt = fileMgr.getInfo(currentSelection[3], false)
flMgr.showTextReader(titleTxt, [bodyTxt])
end if
end

on refreshItems()
showEditItems(editMode)
setNowPlayingItems()
updatePLWindow()
end

on showEditItems(yesno)
if yesno = true then
_movie.sprite["artistInst"].visible = false
_movie.sprite["albumInst"].visible = false
_movie.sprite["trackInst"].visible = false
_movie.sprite["trackNoInst"].visible = false
_movie.sprite["albumArtTextInst"].visible = false
_movie.sprite["genreInst"].visible = false
_movie.sprite["albumArtInst"].visible = false
_movie.sprite["visInst"].visible = false
_movie.sprite["playlistEditTitleInst"].visible = true
_movie.sprite["playlistEditTextInst"].visible = true
else
_movie.sprite["artistInst"].visible = true
_movie.sprite["albumInst"].visible = true
_movie.sprite["trackInst"].visible = true
_movie.sprite["trackNoInst"].visible = true
_movie.sprite["albumArtTextInst"].visible = true
_movie.sprite["genreInst"].visible = true
if artMode = true then
_movie.sprite["albumArtInst"].visible = true
else
_movie.sprite["visInst"].visible = true
end if
_movie.sprite["playlistEditTitleInst"].visible = false
_movie.sprite["playlistEditTextInst"].visible = false
end if
end

on setNowPlayingItems()
--show the now playing items
if not voidP(currentAudioFileTags) then
member("artist").text = currentAudioFileTags["artist"]
member("album").text = currentAudioFileTags["album"]
member("track").text = currentAudioFileTags["title"]

```

```

    trk = currentAudioFileTags["track"]
    if trk >= 1 then
        member("trackNo").text = "Track " & trk
    else
        member("trackNo").text = ""
    end if
    member("genre").text = currentAudioFileTags["genre"]
    member("albumArtText").text = ""
else
    currentSong = plAudio.navGetPlaying()
    if not voidP(currentSong) then
        member("artist").text = fileMgr.getContainingFolder(currentSong[3])
        member("album").text = ""
        member("track").text = fileMgr.getNameNoExt(currentSong[3])
        member("trackNo").text = ""
        member("albumArtText").text = ""
        member("genre").text = ""
        setAlbumArt()
    else
        member("artist").text = ""
        member("album").text = ""
        member("track").text = ""
        member("trackNo").text = ""
        member("albumArtText").text = ""
        member("genre").text = ""
        setAlbumArt()
    end if
end if
--detect art from folder
apath = fileMgr.getPath(currentAudioFile)
art = fileMgr.artReader(apath)
if not voidP(art) then
    if count(art) > 0 then
        setAlbumArt(apath&art[1][2])
        member("albumArtText").text = art[1][1]
    else
        setAlbumArt()
    end if
else
    setAlbumArt()
end if
end

on setAlbumArt(fname)
    --accepts filename -- if no filename received then hides art
    if voidP(fname) then
        _movie.sprite["albumArtInst"].visible = false
        if currentAudioPlayMode = 2 or currentAudioPlayMode = 3 then
            _movie.sprite["visInst"].visible = true
        end if
        artMode = false
    else
        _movie.sprite["visInst"].visible = false
        member("albumArt").filename = fname
        _movie.sprite["albumArtInst"].visible = true
        artMode = true
    end if
end

on navBackKey()
    if editMode = true then
        if plAudio.getPl() <> tempPlaylist then
            flMgr.showMsgBox("Save Changes","Save the changes made to the
playlist?","question",["Yes","No","Keep Editing"],1)
        else
            --there are no changes, dont ask and dont restore
            exitEditMode(true) --told to save as save really means "dont revert to old"
        end if
    else
        --do back stuff (return to previous part of movie)
        if backHistory[the frameLabel] <> "" then _movie.go(_movie.label(backHistory[the frameLabel]))
    end if
end

--playlist manipulation
on navUp()
    if plAudio.navUp() = true then updatePLWindow()
end

on navDown()
    if plAudio.navDown() = true then updatePLWindow()
end

on navPageUp()
    if plAudio.navPageUp() = true then updatePLWindow()
end

on navPageDown()
    if plAudio.navPageDown() = true then updatePLWindow()
end

on navEnter()
    if editMode = true then
        --in edit mode: highlight the item

```

```

    if plAudio.highlight() = true then
        updatePLWindow()
    end if
else
    --in norm node: play the item
    plAudio.navSelect()
    refreshItems()
end if
end

--deleting, moving and sorting

on navDelete()
    if plAudio.del() = true then updatePLWindow()
end

on navMoveUp()
    if plAudio.moveUp() = true then updatePLWindow()
end

on navMoveDown()
    if plAudio.moveDown() = true then updatePLWindow()
end

on randomise()
    if plAudio.randomise() = true then updatePLWindow()
end

on sortAlphabetical()
    if plAudio.sortAlphabetical() = true then updatePLWindow()
end

on navHighlightNone()
    if plAudio.highlightNone() = true then updatePLWindow()
end

--Update fasttext keys
on updateFTMenu()
    pl = plAudio.getPL()
    if editMode = true then
        --show buttons for when in edit mode
        if count(pl) > 0 then
            flMgr.setFTMenu("red", ["Remove"])
            flMgr.setFTMenu("green", ["Select None"])
            flMgr.setFTMenu("yellow", ["Move Down"])
            flMgr.setFTMenu("blue", ["Move Up"])
        end if
    else
        --show normal buttons
        if count(pl) > 0 then
            flMgr.setFTMenu("red", ["Edit Playlist"])
            flMgr.setFTMenu("green", ["File", "Save", "Load", "Clear"])
            flMgr.setFTMenu("yellow", ["Sort", "Randomise", "Alphabetical"])
            flMgr.setFTMenu("blue", ["Repeat Mode", "Repeat Off", "Repeat All", "Repeat Track"])
        else
            flMgr.setFTMenu("green", ["Load"])
        end if
    end if
end

--Update flash playlist
on updatePLWindow(me)
    pl = plAudio.getPL()
    --calc page nos
    noFiles = "No songs"
    noPages = ""
    pages = ceil((count(pl) / 16.0))
    page = 1
    sel = 0
    selOnPage = 0
    totalHighlighted = 0
    if count(pl) > 0 then
        repeat with i = 1 to count(pl)
            if pl[i][4] = 1 then sel = i
            if pl[i][5] = 1 then totalHighlighted = totalHighlighted + 1
        end repeat
    end if
    page = ceil((sel / 16.0))
    noPages = "Page " & page & "/" & pages
    if editMode = true then
        noFiles = sel & " (" & totalHighlighted & " / " & count(pl) & " selected)"
    else
        noFiles = sel & " of " & count(pl)
    end if
    selOnPage = sel - (16*(page-1))
    selString = ",100" --saves numbers of items on screen that should be highlighted in pale yellow
    --make string used to construct an array in the flash object
    flFileString =
    "_movie.sprite["&QUOTE;"playlistInst"&QUOTE;].fileWindow.updateFileWindow(_movie.sprite["&QUOTE;"playlist
    Inst"&QUOTE;].newObject("&QUOTE;"Array"&QUOTE;
    highlightCounter = 1 --keeps track of the current item and if it should be highlighted
    if (page * 16) > count(pl) then
        --there wont be 16 items
        repeat with i = ((page * 16) - 15) to count(pl)
            f = pl.getAt(i)
            iconType = "none"
            if f[6] = 1 then iconType = "play"

```

```

    if f[6] = 2 then iconType = "played" --need to make a PLAYED icon
    if f[6] = 3 then iconType = "error"
    flFileString = flFileString & ",
_movie.sprite["&QUOTE;"playlistInst"&QUOTE;"].newObject("&QUOTE;"Array"&QUOTE;", "&QUOTE;" & iconType
&QUOTE;"&QUOTE;"&QUOTE;" f[1] &QUOTE;")"
    if f[5] = 1 then selString = selString & ", " & highlightCounter-1 --if part of a selection then
add to string
    highlightCounter = highlightCounter + 1
end repeat
else
--the page contains 16 items
repeat with i = ((page * 16) - 15) to (page * 16)
    f = pl.getAt(i)
    iconType = "none"
    if f[6] = 1 then iconType = "play"
    if f[6] = 2 then iconType = "played" --need to make a PLAYED icon
    if f[6] = 3 then iconType = "error"
    flFileString = flFileString & ",
_movie.sprite["&QUOTE;"playlistInst"&QUOTE;"].newObject("&QUOTE;"Array"&QUOTE;", "&QUOTE;" & iconType
&QUOTE;"&QUOTE;"&QUOTE;" f[1] &QUOTE;")"
    if f[5] = 1 then selString = selString & ", " & highlightCounter-1 --if part of a selection then
add to string
    highlightCounter = highlightCounter + 1
end repeat
end if
flFileString = flFileString & ")")"
do flFileString
do "_movie.sprite["&QUOTE;"playlistInst"&QUOTE;"].fileWindow.select(selOnPage -
1, _movie.sprite["&QUOTE;"playlistInst"&QUOTE;"].newObject("&QUOTE;"Array"&QUOTE;" selString &"))"
else
--if no items in playlist

_movie.sprite["playlistInst"].fileWindow.updateFileWindow(_movie.sprite["playlistInst"].newObject("Array")
)
_movie.sprite["playlistInst"].fileWindow.select(0, _movie.sprite["playlistInst"].newObject("Array")) --
select nothing
end if
_movie.sprite["playlistInst"].fileWindow.scrollBar.setScrollBar(page, pages)
_movie.sprite["playlistInst"].noFiles.text = noFiles
_movie.sprite["playlistInst"].noPages.text = noPages
end

```

picture

```

--picture
global backHistory

property firstRun
global plAudio
global flMgr
global fxObj
global fileMgr
global fileTypes
global lastPage --used to prevent thumbnails from reloading if page has not changed

global currentPictureFolder
global currentPicturePath
global currentPictureSelType --so ft menu represents selection

on isOKToAttach (me, aSpriteType, aSpriteNum)
    tIsOk = 0
    if aSpriteType = #script then
        tIsOk = 1
    end if
    return(tIsOk)
end on

--key commands
on keyUp()
    case (getKey()) of
        --add keys relevant to this frame
        "nowPlayingKey": gotoNowPlaying(the framelabel)
        "powerKey": gotoMenu(the framelabel)
        "playKey": playAudio()
        "nextKey": plAudio.nextSong()
        "prevKey": plAudio.prevSong()
        "stopKey": stopAudio()
        "infoKey": navShowInfo()
        "pageUpKey": navPageUp()
        "pageDownKey": navPageDown()
        "upKey": navUp()
        "downKey": navDown()
        "leftKey": navLeft()
        "rightKey": navRight()
        "enterKey": navSelect()
        "backKey": navBack()
        "redKey": flMgr.navFTMenu("red")
        "greenKey": flMgr.navFTMenu("green")
        "yellowKey": flMgr.navFTMenu("yellow")
        "blueKey": flMgr.navFTMenu("blue")
        --"nowPlayingKey":
    end case

```

```

end

on mouseUp()
  click = getMouseClicked()
  case (click[1]) of
    --add flash buttons relevant to frame
  end case
end

on beginSprite(scrollBarInst)
  firstRun = true
  --used to run code when a frame is first entered
  member("title").text = "Media Guide"
end

on endSprite(scrollBarInst)
  --run once on exit
  flMgr.hideFTMenu()
end

on exitFrame(me)
  go the frame
  flEvent = getFlashEvent()
  audioTicker() --keeps song changing/playlist running
  if flEvent <> "" then
    case (flEvent) of
      "yellow_Slideshow": initSlideshow()
      "yellow_View": initViewImage()
    end case
  end if
end

on enterFrame()
  if firstRun = true then
    firstRun = false
    member("title2Line").text = "Picture Browser"
    currentPictureFolder = readFolder(currentPicturePath,true)
    updateSlots()
    updateFTMenu()
  end if
end

--slide show/image initialisation

on initViewImage()
  if currentPictureSelType = "image" then
    sel = getSelection()
    if not voidP(sel) then
      viewImage(currentPicturePath,sel[4], the frameLabel)
    end if
  end if
end

on initSlideshow()
  sel = getSelection()
  if not voidP(sel) then
    folderImages = readFolder(currentPicturePath&sel[4],false) --false- don't include folders
    if count(folderImages) > 0 then
      playSlideShow(currentPicturePath&sel[4], folderImages,the frameLabel)
    else
      flMgr.showMsgBox("Folder Error","There are no images in this folder.,"info",["OK"],1)
    end if
  end if
end

on navShowInfo()
  if currentPictureSelType <> "closed" then
    sel = getSelection()
    if not voidP(sel) then
      titleTxt = "Info for " & sel[1]
      bodyTxt = fileMgr.getInfo(currentPicturePath&sel[4],false)
      flMgr.showTextReader(titleTxt, [bodyTxt])
    end if
  end if
end

--method for reading folders
--this version is hard-coded for image file types

on readFolder(pathStr, includeFolders)
  --does not use general folder reader as requirements are different
  fc = fxObj.fx_FolderToList(pathStr)
  lastPage = 0 --force thumbnails to be redrawn
  folderContents = [] --used at the end as it remains unsorted
  folders = [] --these are separate to ensure folders always come first
  files = [] --files array is always added afterwards
  if not voidP(fc) then
    if count(fc) > 0 then
      repeat with f in fc
        if f contains "\" then
          if includeFolders = true then
            --its a folder
            folders.add([chars(f,1,f.length - 1),"closed",0,f])
          end if
        end if
      end repeat
    end if
  end if
end

```

```

        end if
    else
        if fileMgr.searchList(fileTypes["image"],fileMgr.getExtension(f)) then
            --is an image
            files.add([f,"image",0,f])
        end if
    end if
end repeat
folders.sort() --folders and files lists sorted alphabetically
files.sort() --since director maintains sorted lists, the lists are added to a non-sorted list
repeat with a = 1 to count(folders)
    folderContents.add(folders[a])
end repeat
repeat with a = 1 to count(files)
    folderContents.add(files[a])
end repeat
else
end if
end if
if count(folderContents) > 0 then folderContents[1][3] = 1
return folderContents
end

--navigation commands
--even though navigation is a grid navigation is simple and works on these rules:
--LEFT: -1
--RIGHT: +1
--DOWN: +3
--UP: -3
--PGDN: +9
--PGUP: -9
--updatesSlots draws this as a grid

--navLeft/Right are the same as up/down as they +/- by 1
on navLeft()
    if count(currentPictureFolder) > 0 then
        repeat with n = 1 to count(currentPictureFolder)
            if currentPictureFolder[n][3] = 1 then
                if n = 1 then
                    exit repeat --at top do nothing
                else
                    currentPictureFolder[n][3] = 0
                    currentPictureFolder[n-1][3] = 1 --select item above
                    updateSlots()
                    updateFTMenu()
                    exit repeat
                end if
            end if
        end repeat
    end if
end

on navRight()
    if count(currentPictureFolder) > 0 then
        repeat with n = 1 to count(currentPictureFolder)
            if currentPictureFolder[n][3] = 1 then
                if n = count(currentPictureFolder) then
                    exit repeat -- at bottom do nothing
                else
                    currentPictureFolder[n][3] = 0
                    currentPictureFolder[n+1][3] = 1 --select item above
                    updateSlots()
                    updateFTMenu()
                    exit repeat
                end if
            end if
        end repeat
    end if
end

--nav up/down are the same as pgup/pgdn as this code handles moving by more than 1 space
on navUp()
    if count(currentPictureFolder) > 0 then
        repeat with n = 1 to count(currentPictureFolder)
            if currentPictureFolder[n][3] = 1 then
                if n = 1 then
                    exit repeat -- at top do nothing
                else
                    if (n-3) < 1 then
                        --go to top of list
                        currentPictureFolder[n][3] = 0
                        currentPictureFolder[1][3] = 1
                        updateSlots()
                        updateFTMenu()
                    else
                        --move 9 places up
                        currentPictureFolder[n][3] = 0
                        currentPictureFolder[n-3][3] = 1 --select item above
                    end if
                    updateSlots()
                    updateFTMenu()
                    exit repeat
                end if
            end if
        end repeat
    end if
end

```

```

end repeat
end if
end

on navDown()
  if count(currentPictureFolder) > 0 then
    repeat with n = 1 to count(currentPictureFolder)
      if currentPictureFolder[n][3] = 1 then
        if n = count(currentPictureFolder) then
          exit repeat -- at bottom do nothing
        else
          if (n+3) > count(currentPictureFolder) then
            --go to bottom of list
            currentPictureFolder[n][3] = 0
            currentPictureFolder[count(currentPictureFolder)][3] = 1
            updateSlots()
            updateFTMenu()
          else
            --move 9 places down
            currentPictureFolder[n][3] = 0
            currentPictureFolder[n+3][3] = 1 --select item above
          end if
          updateSlots()
          updateFTMenu()
        end repeat
      end if
    end repeat
  end if
end

on navPageUp()
  if count(currentPictureFolder) > 0 then
    repeat with n = 1 to count(currentPictureFolder)
      if currentPictureFolder[n][3] = 1 then
        if n = 1 then
          exit repeat -- at top do nothing
        else
          if (n-9) < 1 then
            --go to top of list
            currentPictureFolder[n][3] = 0
            currentPictureFolder[1][3] = 1
            updateSlots()
            updateFTMenu()
          else
            --move 9 places up
            currentPictureFolder[n][3] = 0
            currentPictureFolder[n-9][3] = 1 --select item above
          end if
          updateSlots()
          updateFTMenu()
        end repeat
      end if
    end repeat
  end if
end

on navPageDown()
  if count(currentPictureFolder) > 0 then
    repeat with n = 1 to count(currentPictureFolder)
      if currentPictureFolder[n][3] = 1 then
        if n = count(currentPictureFolder) then
          exit repeat -- at bottom do nothing
        else
          if (n+9) > count(currentPictureFolder) then
            --go to bottom of list
            currentPictureFolder[n][3] = 0
            currentPictureFolder[count(currentPictureFolder)][3] = 1
            updateSlots()
            updateFTMenu()
          else
            --move 9 places down
            currentPictureFolder[n][3] = 0
            currentPictureFolder[n+9][3] = 1 --select item above
          end if
          updateSlots()
          updateFTMenu()
        end repeat
      end if
    end repeat
  end if
end

on navSelect()
  sel = getSelection()
  if not voidP(sel) then
    if currentPictureSelType = "closed" then
      --open folder
      currentPicturePath = currentPicturePath & sel[4]
      currentPictureFolder = readFolder(currentPicturePath, true)
      updateSlots()
      updateFTMenu()
    end if
  end if
end

```

```

else if currentPictureSelType = "image" then
  initViewImage()
end if
end if
end

--returns the selected array item
on getSelection()
  if count(currentPictureFolder) > 0 then
    repeat with n = 1 to count(currentPictureFolder)
      if currentPictureFolder[n][3] = 1 then
        return currentPictureFolder[n]
      end if
    end repeat
    return void --if nothing return void
  end if
end

on navBack() --up a level
  currentPicturePath = fileMgr.getUpOneLevel(currentPicturePath)
  currentPictureFolder = readFolder(currentPicturePath, true)
  updateSlots()
  updateFTMenu()
  updateFTMenu()
end

on updateFTMenu()
  flMgr.hideFtMenu()
  if currentPictureSelType <> "" then -- dont show ftMenuItems items if no files
    --change PLAY/QUEUE text as appropriate
    if currentPictureSelType = "closed" then
      --folder when viewing image files
      put "made the folder decision"
      flMgr.setFTMenu("yellow", ["Slideshow"])
    else if currentPictureSelType = "image" then
      --single file viewing images
      flMgr.setFTMenu("yellow", ["View"])
    end if
  end if
end

on updateSlots()
  member("picturePath").text = currentPicturePath
  --calc page nos
  noFiles = "No files"
  noPages = ""
  pages = ceil((count(currentPictureFolder) / 9.0))
  page = 1
  sel = 0
  selOnPage = 0
  if count(currentPictureFolder) > 0 then
    repeat with i = 1 to count(currentPictureFolder)
      if currentPictureFolder[i][3] = 1 then
        sel = i
        --currentSelType = currentFolder[i][2] --sp ft menu knows what options to offer
      end if
    end repeat
    page = ceil((sel / 9.0))
    noPages = "Page " & page & " of " & pages
    if count(currentPictureFolder) > 1 then
      noFiles = "File " & sel & " of " & count(currentPictureFolder) & " selected"
    else
      noFiles = "File 1 of 1 selected"
    end if
    selOnPage = sel - (9*(page-1))
    if lastPage <> page then
      lastPage = page
      if (page * 9) > count(currentPictureFolder) then
        --there wont be 9 items
        counter = 1
        repeat with i = ((page * 9) - 8) to count(currentPictureFolder)
          f = currentPictureFolder.getAt(i)
          if f[2] = "closed" then
            --is a folder
            currentPictureSelType = "closed"
            setSlot(counter, "folder.bmp", f[1])
          else
            --is an image
            currentPictureSelType = "image"
            setSlot(counter, currentPicturePath & f[4], f[1])
          end if
          counter = counter + 1
        end repeat
      if counter < 9 then
        repeat with m = counter to 9
          --clear slots with no image
          setSlot(m, "thumb.bmp", "")
        end repeat
      end if
    else
      counter = 1
      --the page contains 9 items
      repeat with i = ((page * 9) - 8) to (page * 9)

```

```

    f = currentPictureFolder.getAt(i)
    if f[2] = "closed" then
        --is a folder
        currentPictureSelType = "closed"
        setSlot(counter, "folder.bmp",f[1])
    else
        --is an image
        currentPictureSelType = "image"
        setSlot(counter, currentPicturePath & f[4],f[1])
    end if
    counter = counter + 1
end repeat
end if
end if
selectSlot(selOnPage)
else
    resetSlots()
    selectSlot(0)
    currentPictureSelType = ""
    --member("picturePath").text = currentPicturePath
    --currentSelType = "" -- dont offer play/queue up buttons if there are no files
end if
_movie.sprite["scrollBarInst"].scrollBar.setScrollBar(page,pages)
end

on selectSlot(slotNo)
    if not voidP(slotNo) then
        repeat with i = 1 to 9
            if i = slotNo then
                _movie.sprite["slot"&i].forecolor = 4
            else
                _movie.sprite["slot"&i].forecolor = 255
            end if
        end repeat
    end if
end

on setSlot(slotNo,imageFn,textName)
    if not voidP(slotNo) then
        if not voidP(imageFn) then
            if not voidP(textName) then
                member("slot"&slotNo&"Text").text = textName
                member("slot"&slotNo&"Image").filename = imageFn
            end if
        end if
    end if
end

on resetSlots()
    repeat with i = 1 to 9
        member("slot"&i&"Image").filename = "thumb.bmp"
        member("slot"&i&"Text").text = ""
    end repeat
end

```

browser

```

--browser
global backHistory
global plAudio
global plVideo

global fileMgr
global flMgr
global fxObj

global firstRun
global browserFileType --accessed by ses manager to set browsing
global fileTypes
global currentPath --accessed by ses manager to set browsing
global currentFolder
global currentSelType --used for ft menu to decide what options to show

global folderArray --temporary array of files for replace operation

on isOKToAttach (me, aSpriteType, aSpriteNum)
    tIsOk = 0
    if aSpriteType = #script then
        tIsOk = 1
    end if
    return(tIsOk)
end on

--key commands

on keyUp()
    case (getKey()) of
        --add keys relevant to this frame
        "nowPlayingKey": gotoNowPlaying(the framelabel)
        "powerKey": gotoMenu(the framelabel)
        "playKey":playAudio()
        "nextKey":plAudio.nextSong()
    end case
end on

```

```

"prevKey":plAudio.prevSong()
"stopKey":stopAudio()
"delKey":
"upKey": navUp()
"downKey": navDown()
"leftKey": navBack()
"rightKey": navSelect()
"infoKey":navShowInfo()
"enterKey": navSelect()
"backKey": navBack()
"pageUpKey": navPageUp()
"pageDownKey": navPageDown()
"redKey": flMgr.navFTMenu("red")
"greenKey": flMgr.navFTMenu("green")
"yellowKey": flMgr.navFTMenu("yellow")
"blueKey": flMgr.navFTMenu("blue")
end case
end

on mouseUp()
click = getMouseClicked()
case (click[1]) of
--add flash buttons relevant to frame
"menu0": menu0(click[2])
end case
end

--may need to change to a flash obj that will be there from the start!
on beginSprite(fileBrowserInst)
firstRun = true
end

on endSprite(fileBrowserInst)
flMgr.hideFTMenu()
end

on exitFrame(me)
go the frame
flEvent = getFlashEvent()
if flEvent <> "" then
case (flEvent) of
--message boxes
"msgBox_Clear Music Playlist_Yes":plAudio.clearPlaylist()
"msgBox_Clear Video Playlist_Yes":plVideo.clearPlaylist()
"msgBox_Play Music In Folder_Yes": playFolder(true, folderArray)
"msgBox_Play Video In Folder_Yes": playFolder(false, folderArray)
"msgBox_Play Music File_Yes": playTheFile()
"msgBox_Play Video File_Yes": playTheFile()
--view by file type
"red_View all": changeFileType("all")
"red_View music": changeFileType("music")
"red_View video": changeFileType("video")
"red_View pictures": changeFileType("image")
--playlist control
"green_View Music": backHistory["sound"] = the frameLabel
_movie.go(_movie.label("sound"))
"green_View Video": backHistory["video"] = the frameLabel
_movie.go(_movie.label("video"))
"green_Clear Music":flMgr.showMsgBox("Clear Music Playlist","Are you sure you want to clear the
current music playlist?", "question", ["Yes", "No"], 2)
"green_Clear Video":flMgr.showMsgBox("Clear Video Playlist","Are you sure you want to clear the
current video playlist?", "question", ["Yes", "No"], 2)
--folder all files
"yellow_Play music": checkPlayFolder(true)
"yellow_Play video": checkPlayFolder(false)
"yellow_View slideshow": initSlideshow()
"blue_Queue music": queueFolder(true)
"blue_Queue video": queueFolder(false)
--folder image
"yellow_Slideshow": initSlideshow()
--folder music/video
--"yellow_Play Folder": checkPlayFolder() --
--"blue_Queue Folder": queueFolder() --these dont know what to q
--file image
"yellow_View": initViewImage()
--file music/video
"yellow_Play": checkPlayFile()
"blue_Queue Up": queueFile()
end case
end if
end

on enterFrame()
if firstRun = true then
firstRun = false
--run once stuff here
currentSelType = ""
_movie.sprite["fileBrowserInst"].setColours(0,75,255,100,75,255,255,10)
member("title2Line").text = "File Browser"
changeFileType(browserFileType) --used to read folder/init windows
end if
end

on changeFileType(newFileType)
member("subTitle").text = "Viewing " &newFileType& " files"
browserFileType = newFileType

```

```

currentFolder = readFolder(currentPath,browserFileType, true)
updateFlWindow()
updateFTMenu()
end

--folder play/queue methods must be told type of file to add
--file methods detect type from current selection
--play methods check first (if folder contains media/current playlist size >1)

on checkPlayFile()
sel = getSelection()
if not voidP(sel) then
if sel[2] = "music" then
plSize = count(plAudio.getPl())
if plSize > 1 then
flMgr.showMsgBox("Play Music File","The current music playlist contains " & plSize & " songs. Are
you sure you want to clear the playlist and add the file?","question",["Yes","No"],2)
else
playTheFile()
end if
else if sel[2] = "video" then
plSize = count(plVideo.getPl())
if plSize > 1 then
flMgr.showMsgBox("Play Video File","The current video playlist contains " & plSize & " items. Are
you sure you want to clear the playlist and add the file?","question",["Yes","No"],2)
else
playTheFile()
end if
end if
end if
end

on playtheFile()
sel = getSelection()
if not voidP(sel) then
if sel[2] = "music" then
plAudio.clearPlaylist()
queueUp(true,0,currentPath&sel[4])
plAudio.advance()
else if sel[2] = "video" then
plVideo.clearPlaylist()
queueUp(false,0,currentPath&sel[4])
plVideo.advance()
end if
end if
end

--checks length of playlist
--if greater than 1 it will show message box, else it will replace instantly
on checkPlayFolder(avMode) --true= audio, false=video
sel = getSelection()
if not voidP(sel) then
fType = "music"
if avMode = false then fType = "video"--use the type of the selected file
folderArray = readFolder(currentPath&sel[4],fType, false) --false- don't include folders
if count(folderArray) > 0 then
if avMode = true then
--for audio files
plSize = count(plAudio.getPl())
if plSize > 1 then
flMgr.showMsgBox("Play Music In Folder","The current music playlist contains " & plSize & "
songs. Are you sure you want to play the folder?","question",["Yes","No"],2)
else
--only 1 file, replace anyway
playFolder(avMode,folderArray)
end if
else if avMode = false then
--for video files
plSize = count(plVideo.getPl())
if plSize > 1 then
flMgr.showMsgBox("Play Video In Folder","The current video playlist contains " & plSize & "
items. Are you sure you want to play the folder?","question",["Yes","No"],2)
else
--only 1 file, replace anyway
playFolder(avMode,folderArray)
end if
end if
else
if avMode = true then
flMgr.showMsgBox("Folder Error","There are no music files in this folder.,"error",["OK"],1)
else
flMgr.showMsgBox("Folder Error","There are no video files in this folder.,"error",["OK"],1)
end if
end if
end if
end

on playFolder(avMode,folderArray)
if not voidP(avMode) then
sel = getSelection()
if not voidP(sel) then
if avMode = true then
--audio files
plAudio.clearPlaylist()
repeat with i = 1 to count(folderArray)

```

```

        queueUp(avMode,0,currentPath&sel[4]&folderArray[i][4])
    end repeat
    plAudio.advance()
else if avMode = false then
    --video files
    plVideo.clearPlaylist()
    repeat with i = 1 to count(folderArray)
        queueUp(avMode,0,currentPath&sel[4]&folderArray[i][4])
    end repeat
    plVideo.advance()
    --do video advance
end if
end if
end if
end

on queueFile()
    sel = getSelection()
    if not voidP(sel) then
        if sel[2] = "music" then
            queueUp(true,0,currentPath&sel[4])
        else if sel[2] = "video" then
            queueUp(false,0,currentPath&sel[4])
        end if
    end if
end

on queueFolder(avMode) --true= audio, false=video
    sel = getSelection()
    if not voidP(sel) then
        ftype = "music"
        if avMode = false then ftype = "video"--use the type of the selected file
        folderArray = readFolder(currentPath&sel[4],ftype,false) --false- don't include folders
        if count(folderArray) > 0 then
            repeat with i = 1 to count(folderArray)
                queueUp(avMode,0,currentPath&sel[4]&folderArray[i][4])
            end repeat
        else
            if avMode = true then
                flMgr.showMsgBox("Folder Error","There are no music files in this folder.,"error",["OK"],1)
            else
                flMgr.showMsgBox("Folder Error","There are no video files in this folder.,"error",["OK"],1)
            end if
        end if
    end if
end

on initViewImage()
    if currentSelType = "image" then
        sel = getSelection()
        if not voidP(sel) then
            viewImage(currentPath,sel[4],the frameLabel)
        end if
    end if
end

on initSlideshow()
    sel = getSelection()
    if not voidP(sel) then
        ftype = "image"
        folderImages = readFolder(currentPath&sel[4],ftype,false) --false- don't include folders
        if count(folderImages) > 0 then
            playSlideShow(currentPath&sel[4],folderImages,the frameLabel)
        else
            flMgr.showMsgBox("Folder Error","There are no images in this folder.,"error",["OK"],1)
        end if
    end if
end

on openTXTFile()
    --if enter pressed then size information is not shown
    if currentSelType = "text" then
        sel = getSelection()
        if not voidP(sel) then
            bodyTxt = fileMgr.readTXTFile(currentPath&sel[4])
            flMgr.showTextReader(sel[1],[bodyTxt])
        end if
    end if
end

on navShowInfo()
    if currentSelType <> "closed" then
        sel = getSelection()
        if not voidP(sel) then
            titleTxt = "Info for " & sel[1]
            bodyTxt = fileMgr.getInfo(currentPath&sel[4],false)
            if currentSelType = "video" then
                --check for txt info files
                txt = fileMgr.txtReader(currentPath,sel[4])
                txt.add(bodyTxt)
                flMgr.showTextReader(titleTxt,txt)
            else if currentSelType = "text" then
                --for txt files show size + contents
            end if
        end if
    end if
end

```

```

        bodyTxt = bodyTxt & numToChar(13) & "Contents: " & numToChar(13) &
fileMgr.readTXTFile(currentPath&sel[4])
        flMgr.showTextReader(titleTxt, [bodyTxt])
    else
        --normal files just show size information
        flMgr.showTextReader(titleTxt, [bodyTxt])
    end if
end if
end if
end
end

on updateFTMenu()
flMgr.hideFTMenu()
flMgr.setFTMenu("red", ["Change view", "View all", "View music", "View video", "View pictures"])
flMgr.setFTMenu("green", ["Playlist", "View Music", "Clear Music", "View Video", "Clear Video"])
if currentSelType <> "" then -- dont show ftMenuItem items if no files
    --change PLAY/QUEUE text as appropriate
    if currentSelType = "closed" and browserFileType = "all" then
        --folder when viewing all files
        flMgr.setFTMenu("yellow", ["Play Folder", "Play music", "Play video", "View slideshow"])
        flMgr.setFTMenu("blue", ["Queue Folder", "Queue music", "Queue video"])
    else if currentSelType = "closed" and browserFileType = "image" then
        --folder when viewing image files
        flMgr.setFTMenu("yellow", ["Slideshow"])
    else if currentSelType = "closed" and browserFileType = "music" then
        --folder when music files
        flMgr.setFTMenu("yellow", ["Play Music"])
        flMgr.setFTMenu("blue", ["Queue Music"])
    else if currentSelType = "closed" and browserFileType = "video" then
        --folder when video files
        flMgr.setFTMenu("yellow", ["Play Video"])
        flMgr.setFTMenu("blue", ["Queue Video"])
    else if currentSelType = "image" then
        --single file viewing images
        flMgr.setFTMenu("yellow", ["View"])
    else
        --single file viewing music/video
        flMgr.setFTMenu("yellow", ["Play"])
        flMgr.setFTMenu("blue", ["Queue Up"])
    end if
end if
end if
end

on readPath(pathStr)
    --produces string of folder names separated by commas for Flash
    if not voidP(pathStr) then
        save = the itemDelimiter
        the itemDelimiter = "\"
        pathContents = ["", 0]
        if chars(pathStr, pathStr.length, pathStr.length) = "\" then
            --remove end \ if there is one
            pathStr = chars(pathStr, 1, pathStr.length - 1)
        end if
        if chars(pathStr, 1, 2) = "\\\" then
            --is a unc path
            pathContents[2] = 1 --1 represents a network icon
            ps = chars(pathStr, 3, pathStr.length)
            repeat with i = 1 to ps.item.count
                pathContents[1] = pathContents[1] & "," & QUOTE & ps.item[i] & QUOTE
            end repeat
        else
            --is a local/mapped drive
            pathContents[2] = 0 --0 = closed folder icon
            repeat with i = 1 to pathStr.item.count
                pathContents[1] = pathContents[1] & "," & QUOTE & pathStr.item[i] & QUOTE
            end repeat
        end if
        the itemDelimiter = save
        return pathContents
    end if
end

on readFolder(pathStr, fTypes, includeFolders)
    --does not use general folder reader as requirements are different
    --include folders can be disabled for when getting files to queue
    fc = fxObj.fx_FolderToList(pathStr)
    folderContents = [] --used at the end as it remains unsorted
    folders = [] --these are separate to ensure folders always come first
    files = [] --files array is always added afterwards
    if not voidP(fc) then
        if count(fc) > 0 then
            repeat with f in fc
                if f contains "\" then
                    if includeFolders = true then
                        --its a folder
                        folders.add([chars(f, 1, f.length - 1), "closed", 0, f])
                    end if
                else
                    --its a file - but only add it if the extension meets criteria
                    if fileMgr.searchList(fileTypes[fTypes], fileMgr.getExtension(f)) = true then
                        if fileMgr.searchList(fileTypes["music"], fileMgr.getExtension(f)) then
                            --is music
                            files.add([f, "music", 0, f])
                        else if fileMgr.searchList(fileTypes["video"], fileMgr.getExtension(f)) then
                            --is video
                            files.add([f, "video", 0, f])
                        end if
                    end if
                end if
            end repeat
        end if
    end if
end

```

```

else if fileMgr.searchList(fileTypes["image"],fileMgr.getExtension(f)) then
  --is an image
  files.add([f,"image",0,f])
else if fileMgr.searchList(fileTypes["text"],fileMgr.getExtension(f)) then
  --is text
  files.add([f,"text",0,f])
else if fileMgr.searchList(fileTypes["playlist"],fileMgr.getExtension(f)) then
  --is text
  files.add([f,"playlist",0,f])
end if
end if
end if
end repeat
folders.sort() --folders and files lists sorted alphabetically
files.sort() --since director maintains sorted lists, the lists are added to a non-sorted list
repeat with a = 1 to count(folders)
  folderContents.add(folders[a])
end repeat
repeat with a = 1 to count(files)
  folderContents.add(files[a])
end repeat
else
end if
end if
if count(folderContents) > 0 then folderContents[1][3] = 1
return folderContents
end

on navUp()
  if count(currentFolder) > 0 then
    repeat with n = 1 to count(currentFolder)
      if currentFolder[n][3] = 1 then
        if n = 1 then
          exit repeat --at top do nothing
        else
          currentFolder[n][3] = 0
          currentFolder[n-1][3] =1 --select item above
          updateFlWindow()
          updateFTMenu()
          exit repeat
        end if
      end if
    end repeat
  end if
end

on navDown()
  if count(currentFolder) > 0 then
    repeat with n = 1 to count(currentFolder)
      if currentFolder[n][3] = 1 then
        if n = count(currentFolder) then
          exit repeat -- at bottom do nothing
        else
          currentFolder[n][3] = 0
          currentFolder[n+1][3] =1 --select item above
          updateFlWindow()
          updateFTMenu()
          exit repeat
        end if
      end if
    end repeat
  end if
end

on navPageUp()
  if count(currentFolder) > 0 then
    repeat with n = 1 to count(currentFolder)
      if currentFolder[n][3] = 1 then
        if n = 1 then
          exit repeat -- at top do nothing
        else
          if (n-10) < 1 then
            --go to top of list
            currentFolder[n][3] = 0
            currentFolder[1][3] =1
            updateFlWindow()
            updateFTMenu()
          else
            --move 10 places up
            currentFolder[n][3] = 0
            currentFolder[n-10][3] =1 --select item above
          end if
          updateFlWindow()
          updateFTMenu()
          exit repeat
        end if
      end if
    end repeat
  end if
end

on navPageDown()
  if count(currentFolder) > 0 then
    repeat with n = 1 to count(currentFolder)
      if currentFolder[n][3] = 1 then
        if n = count(currentFolder) then

```

```

    exit repeat -- at bottom do nothing
  else
    if (n+10) > count(currentFolder) then
      --go to bottom of list
      currentFolder[n][3] = 0
      currentFolder[count(currentFolder)][3] = 1
      updateFlWindow()
      updateFTMenu()
    else
      --move 10 places down
      currentFolder[n][3] = 0
      currentFolder[n+10][3] = 1 --select item above
    end if
    updateFlWindow()
    updateFTMenu()
  exit repeat
end if
end if
end repeat
end if
end

```

```

--returns the selected array item
on getSelection()
  if count(currentFolder) > 0 then
    repeat with n = 1 to count(currentFolder)
      if currentFolder[n][3] = 1 then
        return currentFolder[n]
      end if
    end repeat
    return void --if nothing return void
  end if
end

```

```

on navSelect()
  sel = getSelection()
  if not voidP(sel) then
    if currentSelType = "closed" then
      --open folder
      currentPath = currentPath & sel[4]
      currentFolder = readFolder(currentPath,browserFileType,true)
      updateFlWindow()
      updateFTMenu()
    else if currentSelType = "image" then
      initViewImage()
    else if currentSelType = "text" then
      openTXTFile()
    end if
  end if
end

```

```

on navBack() --up a level
  currentPath = fileMgr.getUpOneLevel(currentPath)
  currentFolder = readFolder(currentPath,browserFileType,true)
  updateFlWindow()
  updateFTMenu()
end

```

```

on updateFlWindow(me)
  --get string of the path to create tree structure
  pathStr = readPath(currentPath)
  do
    _movie.sprite["&QUOTE;"fileBrowserInst"&QUOTE;"].folderWindow.updateFolderWindow(_movie.sprite["&QUOTE;"fileBrowserInst"&QUOTE;"].newObject("&QUOTE;"Array"&QUOTE;"&pathStr[1]&"), "&pathStr[2]&")
  --calc page nos
  noFiles = "No files"
  noPages = ""
  pages = ceil((count(currentFolder) / 10.0))
  page = 1
  sel = 0
  selOnPage = 0
  if count(currentFolder) > 0 then
    repeat with i = 1 to count(currentFolder)
      if currentFolder[i][3] = 1 then
        sel = i
        currentSelType = currentFolder[i][2] --sp ft menu knows what options to offer
      end if
    end repeat
    page = ceil((sel / 10.0))
    noPages = "Page " & page & " of " & pages
    if count(currentFolder) > 1 then
      noFiles = sel & " of " & count(currentFolder) & " selected"
    else
      noFiles = "1 of 1 selected"
    end if
    selOnPage = sel - (10*(page-1))
    --make string used to construct an array in the flash object
    flFileString =
    _movie.sprite["&QUOTE;"fileBrowserInst"&QUOTE;"].fileWindow.updateFileWindow(_movie.sprite["&QUOTE;"fileBrowserInst"&QUOTE;"].newObject("&QUOTE;"Array"&QUOTE;"
    if (page * 10) > count(currentFolder) then
      --there wont be 10 items
      repeat with i = ((page * 10) - 9) to count(currentFolder)
        f = currentFolder.getAt(i)

```

```

    flFileString = flFileString & ",
    _movie.sprite["&QUOTE&"fileBrowserInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE&", "&QUOTE& f [2]
    &QUOTE&", "&QUOTE& f [1] &QUOTE&")"
    end repeat
  else
    --the page contains 10 items
    repeat with i = ((page * 10) - 9) to (page * 10)
      f = currentFolder.getAt(i)
      flFileString = flFileString & ",
    _movie.sprite["&QUOTE&"fileBrowserInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE&", "&QUOTE& f [2]
    &QUOTE&", "&QUOTE& f [1] &QUOTE&")"
    end repeat
  end if
  flFileString = flFileString & ")"
  do flFileString
    _movie.sprite["fileBrowserInst"].fileWindow.select(selOnPage - 1)
  else
    _movie.sprite["fileBrowserInst"].fileWindow.updateFileWindow(_movie.sprite["fileBrowserInst"].newObject("A
    rray"))
    _movie.sprite["fileBrowserInst"].fileWindow.select(0) --select nothing
    currentSelType = "" -- dont offer play/queue up buttons if there are no files
  end if
  _movie.sprite["fileBrowserInst"].fileWindow.scrollBar.setScrollBar(page,pages)
  _movie.sprite["fileBrowserInst"].noFiles.text = noFiles
  _movie.sprite["fileBrowserInst"].noPages.text = noPages
End

```

slideshow

```

--slideshow
global backHistory

global firstRun
global delayFlag
global delayTime

global slideShowMode --boolean, true if show, false if single image
global showArray
global showPath --paths are not included in the folder arrays so are provided
global currentImage

global FTMenuHidden --boolean states if the menu is hidden
global FTMenuUpTime

global slideTitles --boolean for title
global slideRepeat -- 0=off, 1=on, 2= random
global slideInterval
global slideUpTime

global plAudio
global flMgr
global fileMgr

on isOKToAttach (me, aSpriteType, aSpriteNum)
  tIsOk = 0
  if aSpriteType = #script then
    tIsOk = 1
  end if
  return(tIsOK)
end on

--key commands
on keyUp()
  case (getKey()) of
    --add keys relevant to this frame
    "nowPlayingKey": gotoNowPlaying(the frameLabel)
    "powerKey": gotoMenu(the frameLabel)
    "playKey": playAudio()
    "nextKey": plAudio.nextSong()
    "prevKey": plAudio.prevSong()
    "stopKey": stopAudio()
    "leftKey": navLeft()
    "rightKey": navRight()
    "backKey": if backHistory[the frameLabel] <> "" then _movie.go(_movie.label(backHistory[the
    frameLabel]))
    "redKey": showFTMenu("red") --indirect here so the menu can be brought up first
    "greenKey": showFTMenu("green")
    "yellowKey": showFTMenu("yellow")
    "blueKey": showFTMenu("blue")
  end case
end

on mouseUp()
  click = getMouseClicked()
  case (click[1]) of
    --add flash buttons relevant to frame
    --"menu0": menu0(click[2])
  end case
end

on exitFrame(me)

```

```

go the frame
flEvent = getFlashEvent()
audioTicker() --keeps song changing/playlist running
if flEvent <> "" then
  case (flEvent) of
    --flash events here
    "red_Browse": put "red_Browse"
    "green_Titles Off": slideTitles = true
    flMgr.setFTMenu("green",["Titles On"])
    "green_Titles On":slideTitles = false
    flMgr.setFTMenu("green",["Titles Off"])
    "yellow_Restart Show": navRestart()
    "yellow_Repeat Off": slideRepeat = 0
    "yellow_Repeat On": slideRepeat = 1
    "yellow_Repeat Random": slideRepeat = 2
    "blue_Manual": slideInterval = 0
    "blue_2 seconds": slideInterval = 2000
    "blue_5 seconds": slideInterval = 5000
    "blue_10 seconds": slideInterval = 10000
    "blue_20 seconds": slideInterval = 20000
  end case
end if
--osd auto-hide check
if ticksToMs(the timer - FTMenuUpTime) > (5000) then
  --hide osd
  flMgr.hideFTMenu()
  FTMenuHidden = true
end if
--auto slide advance
if slideInterval <> 0 and slideShowMode = true then
  if (ticksToMs(the timer) - slideUpTime) > (slideInterval) then
    navAdvance()
  end if
end if
end

on beginSprite(scrollBarInst)
  firstRun = true
  --used to run code when a frame is first entered
end

on endSprite(scrollBarInst)
  --run once on exit
  flMgr.hideFTMenu()
end

on enterFrame()
  if firstRun = true then
    firstRun = false
    member("title").text = "Picture Browser"
    FTMenuHidden = true
    --defaults for a new slide show
    slideInterval = 5000
    slideRepeat = 1
    FTMenuUpTime = 0
    if slideShowMode = true then
      navRestart()
    else
      _movie.sprite["slideShowTitleInst"].visible = false
      showSlide(currentImage)
    end if
  end if
end

on showFTMenu(colour)
  if slideShowMode = true then
    if FTMenuHidden = true then
      flMgr.setFTMenu("red",["Browse"])
      if slideRepeat = false then
        flMgr.setFTMenu("green",["Titles Off"])
      else
        flMgr.setFTMenu("green",["Titles On"])
      end if
      flMgr.setFTMenu("yellow",["Repeat","Restart Show","Repeat Off","Repeat On","Repeat Random"])
      flMgr.setFTMenu("blue",["Interval","Manual","2 seconds","5 seconds","10 seconds","20 seconds"])
      FTMenuHidden = false
      FTMenuUpTime = the timer
    else
      flMgr.navFTMenu(colour)
      FTMenuUpTime = the timer --keep the menu up for another 5 seconds
    end if
  end if
end

on navLeft()
  if slideShowMode = true then
    if count(showArray) > 0 then
      if currentImage > 1 then
        currentImage = currentImage - 1
        navGotoSlide(currentImage)
      end if
    end if
  end if
end
end

```

```

on navRight()
  if slideShowMode = true then
    if count(showArray) > 0 then
      if currentImage < count(showArray) then
        currentImage = currentImage + 1
        navGotoSlide(currentImage)
      end if
    end if
  end if
end

on navRestart()
  if slideShowMode = true then
    if count(showArray) > 0 then
      currentImage = 1
      navGotoSlide(currentImage)
    end if
  end if
end

on navAdvance() --automatic advancing of images
  if slideShowMode = true then
    if count(showArray) > 0 then
      if slideRepeat = 0 then
        --sequential mode
        if currentImage < count(showArray) then
          currentImage = currentImage + 1
          navGotoSlide(currentImage)
        end if
      else if slideRepeat = 1 then
        --sequential repeat mode
        if currentImage < count(showArray) then
          currentImage = currentImage + 1
        else
          currentImage = 1
        end if
        navGotoSlide(currentImage)
      else if slideRepeat = 2 then
        --random mode
        rnd = random(count(showArray))
        currentImage = rnd
        navGotoSlide(currentImage)
      end if
    end if
  end if
end

on navGotoSlide(slideNo) --used when advanced using directional keys
  if slideShowMode = true then
    if count(showArray) > 0 then
      if slideTitles = true then
        member("slideShowTitle").text = fileMgr.getNameNoExt(showArray[slideNo][1])
        _movie.sprite["slideShowTitleInst"].visible = true
      else
        member("slideShowTitle").text = ""
        _movie.sprite["slideShowTitleInst"].visible = false
      end if
      showSlide(showPath & showArray[slideNo][4])
      slideUpTime = ticksToMs(the timer)
    end if
  end if
end

on showSlide(fName)
  member("slide").filename = fName
  --detect images aspect ratio
  imageWidth = member("slide").width + 0.0 -- required so not integers
  imageHeight = member("slide").height + 0.0
  imageRatio = (imageHeight / imageWidth)
  if (imageRatio <= (600.0 / 800.0)) then
    --reset locations
    _movie.sprite["slideInst"].locV = 0
    _movie.sprite["slideInst"].locH = 0
    --reset size
    _movie.sprite["slideInst"].width = 800
    _movie.sprite["slideInst"].height = 800 * imageRatio
    --centre image
    gap = ((600 - _movie.sprite["slideInst"].height) + 0.0) / 2
    _movie.sprite["slideInst"].locV = gap
  else
    --reset locations
    _movie.sprite["slideInst"].locV = 0
    _movie.sprite["slideInst"].locH = 0
    --reset size
    imageRatio = (imageWidth/imageHeight)
    _movie.sprite["slideInst"].height = 600
    _movie.sprite["slideInst"].width = 600 * imageRatio
    --centre image
    gap = ((800 - _movie.sprite["slideInst"].width) + 0.0) / 2
    _movie.sprite["slideInst"].locH = gap
  end if
end

```

```

on msToTicks(ms)
    return (ms / 1000.0) * 60.0
end

on ticksToMs(tick)
    return (tick / 60.0) * 1000.0
end

```

preVideo

```

--preWmpVid
global currentVideoPlayer
global currentVideoFile
global plVideo

on isOKToAttach (me, aSpriteType, aSpriteNum)
    tIsOk = 0
    if aSpriteType = #script then
        tIsOK = 1
    end if
    return(tIsOK)
end on

--key commands

on keyUp()
    case (getKey()) of
    end case
end

on mouseUp()
    click = getMouseClicked()
    case (click[1]) of
    end case
end

on enterFrame()
    --ensure OSD components are initialised
    _movie.sprite["osdTopInst"].visible = true
    _movie.sprite["osdTopInst"].ink = 100
    _movie.sprite["osdBottomInst"].visible = true
    _movie.sprite["osdBottomInst"].ink = 100
    if currentVideoPlayer = "real" then
        --realPlayer needs an extra frame to initialise
        member("realVideo").fileName = currentVideoFile
    end if
end

```

video

```

--video
global backHistory

--wmpVid
global firstRun
global fileMgr
global plVideo

--these are set in playManager
global currentVideoPlayer
global currentVideoPlayMode
global currentVideoFile
global elapsedVideoTime --so the flash variable does not need to be checked (slower)

global osd
global osdVideoChange --if the video height was modified by osd
global osdUpStartTime

on isOKToAttach (me, aSpriteType, aSpriteNum)
    tIsOk = 0
    if aSpriteType = #script then
        tIsOK = 1
    end if
    return(tIsOK)
end on

--key commands

on keyUp()
    case (getKey()) of
        --add keys relevant to this frame
        "leftKey": rewVideo()
        "rightKey": ffVideo()
        "playKey": playVideo()
        "stopKey": stopVideo()
        "nextKey": plVideo.nextSong()
        "prevKey": plVideo.prevSong()
        "infoKey": toggleOSD()
    end case
end

```

```

    "backKey": backToVideoMenu()
end case
end

on mouseUp()
    click = getMouseClicked()
    case (click[1]) of
        --add flash buttons relevant to frame
        "menu0": menu0(click[2])
    end case
end

on beginSprite(videoInst)
    firstRun = true
end

on exitFrame(me)
    --update time/progress bar if displayed
    if osd = true then
        if (getElapsedVideoTime() - osdUpStartTime) < (5000) then
            evt = msecToHMS(getElapsedVideoTime())
            if elapsedVideoTime <> evt then
                elapsedVideoTime = evt
                _movie.sprite["osdBottomInst"].time.text.text = evt
            end if
        else
            hideOSD()
        end if
    end if
    go the frame
end

on enterFrame()
    if firstRun = true then
        firstRun = false
        setVideo()
    end if
end

on setVideo()
    currentVideoPlayMode = 2
    if currentVideoPlayer <> "real" then
        member(currentVideoPlayer&"Video").fileName = currentVideoFile
    end if
    vidWidth = member(currentVideoPlayer&"Video").width + 0.0 -- required so not integers
    vidHeight = member(currentVideoPlayer&"Video").height + 0.0
    vidRatio = (vidHeight / vidWidth)
    if (vidRatio <= (600.0 / 800.0)) then
        _movie.sprite["videoInst"].width = 800
        _movie.sprite["videoInst"].height = 800 * vidRatio
    else
        vidRatio = (vidWidth/vidHeight)
        _movie.sprite["videoInst"].height = 600
        _movie.sprite["videoInst"].width = 600 * vidRatio
    end if
    --osd set-up
    osd = false
    osdVideoChange = false
    showOSD()
    if currentVideoPlayer = "real" then
        _movie.sprite["videoInst"].play() --real videos dont automatically play
    end if
end

on playVideo()
    if currentVideoPlayMode = 1 then
        --stop to play"
        currentVideoPlayMode = 2
        if currentVideoPlayer = "qt" then
            _movie.sprite["videoInst"].movieRate = 1
        else
            _movie.sprite["videoInst"].play()
        end if
        showOSD()
    else if currentVideoPlayMode = 2 then
        --play to pause"
        currentVideoPlayMode = 3
        if currentVideoPlayer = "qt" then
            _movie.sprite["videoInst"].movieRate = 0
        else
            _movie.sprite["videoInst"].pause()
        end if
        showOSD()
    else if currentVideoPlayMode = 3 then
        --pause to play"
        currentVideoPlayMode = 2
        if currentVideoPlayer = "qt" then
            _movie.sprite["videoInst"].movieRate = 1
        else
            _movie.sprite["videoInst"].play()
        end if
        showOSD()
    end if
end

```

```

else if currentVideoPlayMode = 0 then
  --no file (back to menu)"
  put "NO VIDEO FILE SET TO PLAY"
  backToVideoMenu()
  -- _movie.go(_movie.label("video"))
end if
end

on stopVideo()
if currentVideoPlayMode = 2 then
  if currentVideoPlayer = "qt" then
    _movie.sprite["videoInst"].currentTime = 0
    _movie.sprite["videoInst"].movieRate = 0
  else
    _movie.sprite["videoInst"].stop()
  end if
  currentVideoPlayMode = 1
  showOSD()
else if currentVideoPlayMode = 1 then
  backToVideoMenu()
end if
end

on backToVideoMenu()
if currentVideoPlayMode = 2 then stopVideo()
--_movie.go(_movie.label("video"))
put "STOPPING VIDEO"
_movie.go(_movie.label("postVid")) --ensures flash sprites are properly initialised
end

on ffVideo()
if currentVideoPlayMode = 2 then
_movie.sprite["videoInst"].currentTime = _movie.sprite["videoInst"].currentTime + 20000
showOSD()
end if
end

on rewVideo()
if currentVideoPlayMode = 2 then
_movie.sprite["videoInst"].currentTime = _movie.sprite["videoInst"].currentTime - 20000
showOSD()
end if
end

on getElapsedVideoTime()
return integer(_movie.sprite["videoInst"].currentTime)
end

on getVideoDuration
if currentVideoPlayer = "qt" then
return integer(ticksToMs(_movie.sprite["videoInst"].duration))
else
return integer(_movie.sprite["videoInst"].duration)
end if
end

on showOSD()
osdUpStartTime = getElapsedVideoTime()
if osd = false then
if (_movie.sprite["videoInst"].height > 560) then
osdVideoChange = true
_movie.sprite["videoInst"].height = _movie.sprite["videoInst"].height - 120
end if
osd = true
--show/update OSD
_movie.sprite["osdTopInst"].visible = true
_movie.sprite["osdTopInst"].ink = 100
_movie.sprite["osdBottomInst"].visible = true
_movie.sprite["osdBottomInst"].ink = 100
_movie.sprite["osdTopInst"].setColours(0,75,255,100)
_movie.sprite["osdBottomInst"].setColours(0,75,255,100)
_movie.sprite["osdTopInst"].titleBox.text.text = fileMgr.getNameNoExt(currentVideoFile)
end if
--if OSD is already up just update the play mode display only
if currentVideoPlayMode = 1 then
_movie.sprite["osdTopInst"].playbackIcon.setIcon("stop")
else if currentVideoPlayMode = 2 then
_movie.sprite["osdTopInst"].playbackIcon.setIcon("play")
else if currentVideoPlayMode = 3 then
_movie.sprite["osdTopInst"].playbackIcon.setIcon("pause")
end if
end if
end

on hideOSD()
if osd = true then
if (osdVideoChange = true) then
_movie.sprite["videoInst"].height = _movie.sprite["videoInst"].height + 120
osdVideoChange = false
end if
_movie.sprite["osdTopInst"].visible = false
_movie.sprite["osdTopInst"].ink = 0
_movie.sprite["osdBottomInst"].visible = false
_movie.sprite["osdBottomInst"].ink = 0
end if
end

```

```

    osd = false
  end if
end

on toggleOSD()
  if (osd = false) then
    showOSD()
  else
    hideOSD()
  end if
end

```

videoPl

```

--sound
global backHistory

global flMgr
global plVideo
global fileMgr
global repeatMode

global firstRun
global editMode

global tempPlaylist -- restored if the modifications are discarded

on isOKToAttach (me, aSpriteType, aSpriteNum)
  tIsOk = 0
  if aSpriteType = #script then
    tIsOk = 1
  end if
  return(tIsOk)
end on

--key commands
on keyUp()
  case (getKey()) of
    --add keys relevant to this frame
    "nowPlayingKey": plVideo.advance()
    "powerKey": gotoMenu(the framelabel)
    "infoKey": navShowInfo()
    "playKey": plVideo.advance()
    "nextKey":
    "prevKey":
    "stopKey":
    "upKey": navUp()
    "downKey": navDown()
    "leftKey": rewAudio()
    "rightKey": ffAudio()
    "enterKey": navEnter()
    "backKey": navBackKey()
    "redKey": flMgr.navFTMenu("red")
    "greenKey": flMgr.navFTMenu("green")
    "yellowKey": flMgr.navFTMenu("yellow")
    "blueKey": flMgr.navFTMenu("blue")
    "pageUpKey": navPageUp()
    "pageDownKey": navPageDown()
  end case
end

on mouseUp()
  click = getMouseClicked()
  case (click[1]) of
    --add flash buttons relevant to frame
    "menu0":
  end case
end

on beginSprite(playlistInst)
  firstRun = true
end

on endSprite(playlistInst)
  flMgr.hideFTMenu()
end

on exitFrame(me)
  go the frame
  --icon update
  flEvent = getFlashEvent()
  if flEvent <> "" then
    case (flEvent) of
      --view by file type
      --edit mode
      "red_Remove": navDelete()
      "green_Select None": navHighlightNone()
      "yellow_Move Down": navMoveDown()
      "blue_Move Up": navMoveUp()
      "blue_Move Down": navMoveUp()
      --normal mode
      "red_Edit Playlist": setEditMode()
      "green_Save":
    end case
  end if
end

```



```

on showEditItems(yesno)
  if yesno = true then
    --show edit items
    _movie.sprite["playlistEditTitleInst"].visible = true
    _movie.sprite["playlistEditTextInst"].visible = true
    --hide description items
    _movie.sprite["videoInfoTextInst"].visible = false
    _movie.sprite["videoInfoTitleInst"].visible = false
    _movie.sprite["videoInfoVTitle"].visible = false
  else
    --vice versa
    _movie.sprite["playlistEditTitleInst"].visible = false
    _movie.sprite["playlistEditTextInst"].visible = false
    _movie.sprite["videoInfoTextInst"].visible = true
    _movie.sprite["videoInfoTitleInst"].visible = true
    _movie.sprite["videoInfoVTitle"].visible = true
  end if
end

on navBackKey()
  if editMode = true then
    if plVideo.getPl() <> tempPlaylist then
      flMgr.showMsgBox("Save Changes","Save the changes made to the
playlist?","question",["Yes","No","Keep Editing"],1)
    else
      --there are no changes, dont ask and dont restore
      exitEditMode(true) --told to save as save really means "dont revert to old"
    end if
  else
    --do back stuff (return to previous part of movie)
    if backHistory[the frameLabel] <> "" then _movie.go(_movie.label(backHistory[the frameLabel]))
  end if
end

--playlist manipulation
on navUp()
  if plVideo.navUp() = true then
    updatePLWindow()
    setInfo()
  end if
end

on navDown()
  if plVideo.navDown() = true then
    updatePLWindow()
    setInfo()
  end if
end

on navPageUp()
  if plVideo.navPageUp() = true then
    updatePLWindow()
    setInfo()
  end if
end

on navPageDown()
  if plVideo.navPageDown() = true then
    updatePLWindow()
    setInfo()
  end if
end

on navEnter()
  if editMode = true then
    --in edit mode: highlight the item
    if plVideo.highlight() = true then
      updatePLWindow()
    end if
  else
    --in norm mode: play the item
    plVideo.navSelect() --am i causing crashes --doesnt seem like it
    --plVideo.advance()
  end if
end

--deleting, moving and sorting
on navDelete()
  if plVideo.del() = true then updatePLWindow()
end

on navMoveUp()
  if plVideo.moveUp() = true then updatePLWindow()
end

on navMoveDown()
  if plVideo.moveDown() = true then updatePLWindow()
end

on randomise()
  if plVideo.randomise() = true then updatePLWindow()
end

on sortAlphabetical()
  if plVideo.sortAlphabetical() = true then updatePLWindow()
end

```

```

end

on navHighlightNone()
  if plVideo.highlightNone() = true then updatePlWindow()
end

--Update fasttext keys

on updateFTMenu()
  pl = plVideo.getPL()
  if editMode = true then
    --show buttons for when in edit mode
    if count(pl) > 0 then
      flMgr.setFTMenu("red", ["Remove"])
      flMgr.setFTMenu("green", ["Select None"])
      flMgr.setFTMenu("yellow", ["Move Down"])
      flMgr.setFTMenu("blue", ["Move Up"])
    end if
  else
    --show normal buttons
    if count(pl) > 0 then
      flMgr.setFTMenu("red", ["Edit Playlist"])
      flMgr.setFTMenu("green", ["File", "Save", "Load", "Clear"])
      flMgr.setFTMenu("yellow", ["Sort", "Randomise", "Alphabetical"])
      flMgr.setFTMenu("blue", ["Repeat Mode", "Repeat Off", "Repeat All", "Repeat Track"])
    else
      flMgr.setFTMenu("green", ["Load"])
    end if
  end if
end

--Update flash playlist

on updatePlWindow(me)
  pl = plVideo.getPL()
  --calc page nos
  noFiles = "No videos"
  noPages = ""
  pages = ceil((count(pl) / 16.0))
  page = 1
  sel = 0
  selOnPage = 0
  totalHighlighted = 0
  if count(pl) > 0 then
    repeat with i = 1 to count(pl)
      if pl[i][4] = 1 then sel = i
      if pl[i][5] = 1 then totalHighlighted = totalHighlighted + 1
    end repeat
    page = ceil((sel / 16.0))
    noPages = "Page " & page & "/" & pages
    if editMode = true then
      noFiles = sel & " (" & totalHighlighted & " / " & count(pl) & " selected)"
    else
      noFiles = sel & " of " & count(pl)
    end if
    selOnPage = sel - (16*(page-1))
    selString = ",100" --saves numbers of items on screen that should be highlighted in pale yellow
    --make string used to construct an array in the flash object
    flFileString =
    "_movie.sprite["&QUOTE&"playlistInst"&QUOTE&"].fileWindow.updateFileWindow(_movie.sprite["&QUOTE&"playlist
Inst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE&
highlightCounter = 1 --keeps track of the current item and if it should be highlighted
    if (page * 16) > count(pl) then
      --there wont be 16 items
      repeat with i = ((page * 16) - 15) to count(pl)
        f = pl.getAt(i)
        iconType = "none"
        if f[6] = 1 then iconType = "play"
        if f[6] = 2 then iconType = "played" --need to make a PLAYED icon
        if f[6] = 3 then iconType = "error"
        flFileString = flFileString & ",
    _movie.sprite["&QUOTE&"playlistInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE&,"&QUOTE& iconType
&QUOTE&","&QUOTE& f[1] &QUOTE&")"
        if f[5] = 1 then selString = selString & "," & highlightCounter-1 --if part of a selection then
add to string
        highlightCounter = highlightCounter + 1
      end repeat
    else
      --the page contains 16 items
      repeat with i = ((page * 16) - 15) to (page * 16)
        f = pl.getAt(i)
        iconType = "none"
        if f[6] = 1 then iconType = "play"
        if f[6] = 2 then iconType = "played" --need to make a PLAYED icon
        if f[6] = 3 then iconType = "error"
        flFileString = flFileString & ",
    _movie.sprite["&QUOTE&"playlistInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE&,"&QUOTE& iconType
&QUOTE&","&QUOTE& f[1] &QUOTE&")"
        if f[5] = 1 then selString = selString & "," & highlightCounter-1 --if part of a selection then
add to string
        highlightCounter = highlightCounter + 1
      end repeat
    end if
    flFileString = flFileString & ")")"
    do flFileString
    do "_movie.sprite["&QUOTE&"playlistInst"&QUOTE&"].fileWindow.select(selOnPage -

```

```

1, _movie.sprite["&QUOTE&"playlistInst"&QUOTE&"].newObject("&QUOTE&"Array"&QUOTE& selString &")) "
else
  --if no items in playlist
)
_movie.sprite["playlistInst"].fileWindow.updateFileWindow(_movie.sprite["playlistInst"].newObject("Array")
)
  _movie.sprite["playlistInst"].fileWindow.select(0, _movie.sprite["playlistInst"].newObject("Array")) --
select nothing
end if
  _movie.sprite["playlistInst"].fileWindow.scrollBar.setScrollBar(page, pages)
  _movie.sprite["playlistInst"].noFiles.text = noFiles
  _movie.sprite["playlistInst"].noPages.text = noPages
end

```

preStart

```

--preStart

global flMgr

on isOKToAttach(me, aSpriteType, aSpriteNum)
  tIsOk = 0
  if aSpriteType = #script then
    tIsOK = 1
  end if
  return(tIsOK)
end on

--key commands

on keyUp()
  case (getKey()) of
  end case
end

on mouseUp()
  click = getMouseClicked()
  case (click[1]) of
  end case
end

on enterFrame()
  flMgr.initTextReader()
  flMgr.initMsgBox()
  flMgr.initFTMenu()
  initMediaPlayers()
  initVideoPlayers()
  _movie.sprite["prdWindowInst"].visible = true
end

```

start

```

--start

global flMgr

on isOKToAttach (me, aSpriteType, aSpriteNum)
  tIsOk = 0
  if aSpriteType = #script then
    tIsOK = 1
  end if
  return(tIsOK)
end on

--key commands

on keyUp()
  case (getKey()) of
  end case
end

on mouseUp()
  click = getMouseClicked()
  case (click[1]) of
  end case
end

on enterFrame()
  --run once before movie stuff here
  stopMediaPlayers()
  flMgr.hideFTMenu()
  flMgr.hideTextReader()
  flMgr.hideMsgBox()
  _movie.sprite["prdWindowInst"].visible = false
end

```

--

Flash ActionScript**mainmenu.fla**

```

var currentMenu;
//keeps track of the menu position
//builds the menu instances
//
function initMenuSystem(edgeRGB, centreRGB, mStructure) {
    _root.menuStructure = mStructure;
    for (var i = 0; i<_root.menuStructure.length; i++) {
        _root.attachMovie("menu", "menu"+i, _root.getNextHighestDepth());
        currentItem = _root["menu"+i];
        currentItem.setupMenu = setupMenu;
        currentItem.setupMenu(edgeRGB, centreRGB, _root.menuStructure[i]);
    }
    _root.currentMenu = 0;
}
//
//navigates the menu
//
function viewMenu(menu) {
    if (menu == 0) {
        //hide all submenus
        for (var i = 1; i<menuStructure.length; i++) {
            _root["menu"+i].animate("fadeout");
        }
        if (_root.currentMenu == 0) {
            _root.menu0.animate("fadein");
        } else {
            _root.menu0.animate("enlarge");
        }
        _root.currentMenu = 0;
    } else {
        _root.menu0.animate("shrink");
        _root["menu"+menu].animate("fadein");
        _root.currentMenu = menu;
    }
}
function navMenu(menuNo, itemNo) {
    _root["menu"+menuNo].select(itemNo);
}
//
//function to set upeach menu and add the buttons
//
function setupMenu(edgeRGB, centreRGB, menuItems) {
    this._visible = false;
    //store button text/colours in menu
    this.menuList = menuItems;
    this.origEdgeRGB = edgeRGB;
    this.origCentreRGB = centreRGB;
    //dynamically assign functions
    this.transparency = transparency;
    this.animate = animate;
    this.select = select;
    //set display variables
    this.transAmount = 100;
    this.fadeAmount = 100;
    //add buttons
    for (var i = 0; i<this.menuList.length; i++) {
        this.attachMovie("menuButton", "menuButton"+i, i);
        currentItem = this["menuButton"+i];
        currentItem.text.text = this.menuList[i][0];
        currentItem.selEdgeRGB = this.menuList[i][1];
        currentItem.selCentreRGB = this.menuList[i][2];
        currentItem.setColours = setColours;
        currentItem._x = 0;
        if (i == 0) {
            currentItem._y = 0;
            //if the first item then add it to top left
            currentItem.setColours(currentItem.selEdgeRGB, currentItem.selCentreRGB);
        } else {
            currentItem._y = (this["menuButton"+(i-1)]._y+this["menuButton"+(i-1)]._height+4);
            currentItem.setColours(this.origEdgeRGB, this.origCentreRGB);
        }
    }
    //centering MUST be performed at the end as the clip width changes when elements are added
    this._x = ((Stage.width/2)-(this._width/2));
    this._y = ((Stage.height/2)-(this._height/2));
    this.animate("hide");
}
//
//functions for movie clip appearance - dynamically assigned to each movie clip
//

```

```

function setColours(edgeRGB, centreRGB) {
    edgeTransform = {rb:edgeRGB[0], gb:edgeRGB[1], bb:edgeRGB[2], aa:edgeRGB[3]};
    centreTransform = {rb:centreRGB[0], gb:centreRGB[1], bb:centreRGB[2], aa:centreRGB[3]};
    be = new Color(this.buttonEdge);
    bc = new Color(this.buttonCentre);
    be.setTransform(edgeTransform);
    bc.setTransform(centreTransform);
}
function transparency(transPercent) {
    //sets transparency of menu items (pass it 0-100)
    tf = new Color(this);
    transform = {aa:transPercent};
    tf.setTransform(transform);
}
function animate(trans) {
    this.transition = trans;
    this.gotoAndPlay(2);
}
function select(itemNo) {
    for (var i = 0; i<this.menuList.length; i++) {
        currentItem = this["menuItem"+i];
        if (i == itemNo) {
            currentItem.setColours(currentItem.selEdgeRGB, currentItem.selCentreRGB);
        } else {
            currentItem.setColours(this.origEdgeRGB, this.origCentreRGB);
        }
    }
}
}

```

fasttext.fla

```

var MENUPOS = 1;
//TEST VARIABLE FOR MENU POSITION (NORMALLY BY DIRECTOR)
//assign functions to movie clips
hideAll();
redMenu.setupMenu = setupMenu;
greenMenu.setupMenu = setupMenu;
yellowMenu.setupMenu = setupMenu;
blueMenu.setupMenu = setupMenu;

//functions
function setupMenu(edgeRGB, centreRGB, menuItems) {
    this.menuList = menuItems;
    this.setColours = setColours;
    this.hideMe = hideMe;
    this.showMe = showMe;
    this.setColours(edgeRGB, centreRGB);
    this.menuButton.text.text = this.menuList[0];
    this.select = select;
    if (this.menuList[1] != null) {
        for (var i = 1; i<this.menuList.length; i++) {
            this.attachMovie("menuItem", "menuItem"+i, i);
            currentItem = this["menuItem"+i];
            currentItem.setHighlight = setHighlight;
            currentItem.text.text = this.menuList[i];
            currentItem._x = 0;
            //centre on stage
            //set-up spacing (4) - works backward to put the first item at the top
            currentItem._y = (0-(this.menuButton._height/2))-
            ((currentItem._height+4)*(this.menuList.length-i));
            //add the select function to highlight menuItems
        }
        this.close();
        //menu is never open to begin with
        this.menuItem1.setHighlight(true);
        //first option always highlighted yellow
    }
    this._visible = true;
    //remarked so that all 4 are set-up as dummy menus but cant be seen
}
function setColours(edgeRGB, centreRGB) {
    edgeTransform = {rb:edgeRGB[0], gb:edgeRGB[1], bb:edgeRGB[2], aa:edgeRGB[3]};
    centreTransform = {rb:centreRGB[0], gb:centreRGB[1], bb:centreRGB[2], aa:centreRGB[3]};
    be = new Color(this.menuButton.buttonEdge);
    bc = new Color(this.menuButton.buttonCentre);
    pe = new Color(this.popup.popupEdge);
    pc = new Color(this.popup.popupCentre);
    be.setTransform(edgeTransform);
    bc.setTransform(centreTransform);
    pe.setTransform(edgeTransform);
    pc.setTransform(centreTransform);
}
function setHighlight(boolean) {
    //added to each menuItem
    if (boolean == true) {

```

```

        this.text.backgroundColor = 0xFFFF00;
        this.text.background = true;
    } else {
        this.text.background = false;
    }
}
function select(itemNo) {
    for (var i = 1; i<this.menuList.length; i++) {
        currentItem = this["menuItem"+i];
        if (i == itemNo) {
            currentItem.setHighlight(true);
        } else {
            currentItem.setHighlight(false);
        }
    }
}
function hideMe() {
    this.close();
    this._visible = false;
}
function showMe() {
    this._visible = true;
}
function hideAll() {
    redMenu._visible = false;
    greenMenu._visible = false;
    yellowMenu._visible = false;
    blueMenu._visible = false;
}
}

```

```
//menu code
```

```

var menuList;
var speed = 15;
//no. pixels menu moves each time
var height = marker._y;
//menu height in pixels
function open() {
    //menuitems are not visible here as the menu is yet to animate
    gotoAndPlay(4);
}
function close() {
    //menuitems are hidden before animation takes place
    for (var i = 1; i<menuList.length; i++) {
        currentItem = this["menuItem"+i];
        currentItem._visible = false;
    }
    gotoAndPlay(6);
}
}

```

videoOSDtop fla

```

function setColours(cr, cg, cb, ct) {
    //_root.edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    _root.centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    tb = new Color(_root.titleBox);
    pi = new Color(_root.playbackIcon);
    vi = new Color(_root.videoIcon);
    tb.setTransform(_root.centreTransform);
    pi.setTransform(_root.centreTransform);
    vi.setTransform(_root.centreTransform);
}

```

videoOSDbottom fla

```

progress = 0;

stop();
bl.onRelease = function() {
    time.text.text = "00:05";
    progress++;
    progressBar.setProgress(progress)
}

function setColours(cr, cg, cb, ct) {
    //_root.edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    _root.centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    ai = new Color(_root.arrowIcons);
    ti = new Color(_root.time);
    be = new Color(_root.progressBar);
    ai.setTransform(_root.centreTransform);
    ti.setTransform(_root.centreTransform);
    be.setTransform(_root.centreTransform);
}

```

```

}

//progress bar
stop();

function setProgress(percent){
    x = progressBarBar._x;
    if(percent > 100){percent = 100;}
    this.progressBarBar._xscale = percent;
    this.progressBarBar._x = x;
    //trace(percent);
}

```

clock fla

```

watchDate = new Date();
hourHand._rotation = watchDate.getHours()*30+(watchDate.getMinutes()/2);
minuteHand._rotation = watchDate.getMinutes()*6+(watchDate.getSeconds()/10);

```

```

watchDate = new Date();
var day;
var month;

if (watchDate.getDay()==0){day = "Sunday"}
if (watchDate.getDay()==1){day = "Monday"}
if (watchDate.getDay()==2){day = "Tuesday"}
if (watchDate.getDay()==3){day = "Wednesday"}
if (watchDate.getDay()==4){day = "Thursday"}
if (watchDate.getDay()==5){day = "Friday"}
if (watchDate.getDay()==6){day = "Saturday"}

if (watchDate.getMonth()==0){month = "January"}
if (watchDate.getMonth()==1){month = "February"}
if (watchDate.getMonth()==2){month = "March"}
if (watchDate.getMonth()==3){month = "April"}
if (watchDate.getMonth()==4){month = "May"}
if (watchDate.getMonth()==5){month = "June"}
if (watchDate.getMonth()==6){month = "July"}
if (watchDate.getMonth()==7){month = "August"}
if (watchDate.getMonth()==8){month = "September"}
if (watchDate.getMonth()==9){month = "October"}
if (watchDate.getMonth()==10){month = "November"}
if (watchDate.getMonth()==11){month = "December"}
dateBox.text = day + ", " + watchDate.getDate() + " " + month;

```

fileBrowser fla

```

_root.folderWindow.updateFolderWindow = updateFolderWindow;
_root.fileWindow.updateFileWindow = updateFileWindow;
_root.fileWindow.scrollBar.setScrollBar = setScrollBar;
_root.fileWindow.scrollBar.gotoPercent = gotoPercent;
//colour settings
function setColours(er, eg, eb, et, cr, cg, cb, ct) {
    _root.edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    _root.centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    fie = new Color(_root.fileWindow.fileWindowEdge);
    foe = new Color(_root.folderWindow.folderWindowEdge);
    fib = new Color(_root.fileWindow.fileWindowBack);
    fob = new Color(_root.folderWindow.folderWindowBack);
    fie.setTransform(_root.edgeTransform);
    foe.setTransform(_root.edgeTransform);
    fib.setTransform(_root.centreTransform);
    fob.setTransform(_root.centreTransform);
}
//
function updateFolderWindow(folders, network) {
    this.folder0._visible = false;
    this.folder1._visible = false;
    this.folder2._visible = false;
    this.folder0.setIcon = setIcon;
    this.folder1.setIcon = setIcon;
    this.folder2.setIcon = setIcon;
    this.folder0.setIcon("closed");
    this.folder1.setIcon("closed");
    this.folder2.setIcon("closed");
    this.tree1._visible = false;
    this.tree2._visible = false;
    if (folders.length<4) {
        for (i=0; i<folders.length; i++) {
            if (folders[i] != null) {
                this["folder"+i].text.text = folders[i];
                trace("my i "+i);
                this["tree"+i]._visible = true;
            }
        }
    }
}

```

```

        //there is no tree 0 but flash will ignore
        this["folder"+i]._visible = true;
    }
}
this["folder"+(folders.length-1)].setIcon("open");
if (network == 0) {
    this["folder"+0].setIcon("closed");
} else {
    this["folder"+0].setIcon("network");
}
} else {
    for (i=0; i<3; i++) {
        this["folder"+(2-i)].text.text = folders[((folders.length-1)-i)];
        //work backwards through array
        this["folder"+i]._visible = true;
        this["tree"+i]._visible = true;
        this.folder2.setIcon("open");
    }
}
}
function updateFileWindow(files) {
    this.files = files;
    this.select = select;
    for (i=0; i<10; i++) {
        if (this["element"+i] != null) {
            //if a previous item is there hide it
            this["element"+i]._visible = false;
        }
    }
    for (i=0; i<files.length; i++) {
        if (this["element"+i] == null) {
            //if the items are already there dont make new ones
            this.attachMovie("fileElement", "element"+i, this.getNextHighestDepth());
        }
        currentItem = this["element"+i];
        currentItem._visible = true;
        currentItem.setHighlight = setHighlight;
        currentItem.setIcon = setIcon;
        currentItem.setIcon(files[i][0]);
        currentItem.text.text = files[i][1];
        currentItem._x = 40;
        if (i == 0) {
            currentItem._y = 20;
            currentItem.setHighlight(true);
        } else {
            currentItem._y = (this["element"+(i-1)]._y+this["element"+(i-1)]._height+4);
        }
    }
}
function setIcon(icon) {
    if (icon == "closed") {
        this.gotoAndStop(1);
    }
    if (icon == "open") {
        this.gotoAndStop(2);
    }
    if (icon == "music") {
        this.gotoAndStop(3);
    }
    if (icon == "video") {
        this.gotoAndStop(4);
    }
    if (icon == "image") {
        this.gotoAndStop(5);
    }
    if (icon == "text") {
        this.gotoAndStop(6);
    }
    if (icon == "playlist") {
        this.gotoAndStop(7);
    }
    if (icon == "network") {
        this.gotoAndStop(8);
    }
}
function setHighlight(boolean) {
    //added to each fileElement
    if (boolean == true) {
        this.text.backgroundColor = 0xFFFF00;
        this.text.background = true;
    } else {
        this.text.background = false;
    }
}
function select(itemNo) {

```

```

        trace(itemNo);
        for (var i = 0; i<this.files.length; i++) {
            currentItem = this["element"+i];
            if (i == itemNo) {
                currentItem.setHighlight(true);
            } else {
                currentItem.setHighlight(false);
            }
        }
    }
}
//scroll bar methods
function gotoPercent(percent) {
    p = percent;
    if (p>100) {
        p = 100;
    }
    this.scrollArea.scrollButton._y = (((this.scrollArea.scrollAlpha._height-
this.scrollArea.scrollButton._height)/100)*p);
}
function setScrollBar(currentPage, totalPages) {
    this.scrollArea.scrollButton._height = (this.scrollArea.scrollAlpha._height/totalPages);
    this.scrollArea.scrollButton._x = 0;
    this.scrollArea.scrollButton._y = (this.scrollArea.scrollButton._height*currentPage)-
this.scrollArea.scrollButton._height;
}
}

```

textWindow fla

```

_root.scrollBar.setScrollBar = setScrollBar;
_root.scrollBar.gotoPercent = gotoPercent;
_root.textClip.scrollPercent = scrollPercent;
_root.textClip.scrollUp = scrollUp;
_root.textClip.scrollDown = scrollDown;
_root.textClip.pageUp = pageUp;
_root.textClip.pageDown = pageDown;
_root.textClip.calcPageNos = calcPageNos;
_root.textClip.textSetup = textSetup;
//
function showLastButton(truefalse) {
    if (truefalse == "true") {
        _root.lastButton._visible = true;
    } else {
        _root.lastButton._visible = false;
    }
}
function showNextButton(truefalse) {
    if (truefalse == "true") {
        _root.nextButton._visible = true;
    } else {
        _root.nextButton._visible = false;
    }
}
//setting colours dynamically --no longer uses array as harder to pass to from Director
function setColours(er, eg, eb, et, cr, cg, cb, ct) {
    edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    bo = new Color(_root.border);
    bt = new Color(_root.backTop);
    bm = new Color(_root.backMiddle);
    bb = new Color(_root.backBottom);
    bo.setTransform(edgeTransform);
    bt.setTransform(centreTransform);
    bm.setTransform(centreTransform);
    bb.setTransform(centreTransform);
}
//
//Note: (this.text.bottomScroll-this.text.scroll+1) means lines displayed
//
function scrollPercent(percent) {
    //accepts percentage to scroll to section of document
    this.text.scroll = int((this.text.maxscroll/100)*percent);
    this.calcPageNos();
}
function scrollUp() {
    this.text.scroll--;
    this.calcPageNos();
}
function scrollDown() {
    this.text.scroll++;
    this.calcPageNos();
}
function pageUp() {
    this.text.scroll = this.text.scroll-(this.text.bottomScroll-this.text.scroll+1);
}

```

```

    this.calcPageNos();
}
function pageDown() {
    //+1 prevents last line from displaying at the top again
    //although desirable in windows it makes page number display appear erratic
    this.text.scroll = this.text.bottomScroll+1;
    this.calcPageNos();
}
function calcPageNos() {
    currentPage = Math.ceil(this.text.scroll/(this.text.bottomScroll-this.text.scroll+1));
    totalPages = Math.ceil(this.tLines/(this.text.bottomScroll-this.text.scroll+1));
    _root.pageNoDisplay.text = "Page "+currentPage+" of "+totalPages;
    _root.scrollBar.setScrollBar(currentPage, totalPages);
}
function textSetup(textToDisplay) {
    this.text._visible = false;
    this.text.wordWrap = true;
    this.text.multiline = true;
    this.text.selectable = false;
    this.text.text = textToDisplay;
    this.text.scroll = this.text.maxscroll;
    //scrolls the text box to its max so the
    this.tLines = this.text.bottomScroll;
    //bottom line is displayed (and its no is recorded)
    i = this.tLines;
    while (i>=(this.text.bottomScroll-this.text.scroll+1)) {
        //scroll = current line shown
        i = i-(this.text.bottomScroll-this.text.scroll+1);
    }
    if (i>0) {
        while (i<(this.text.bottomScroll-this.text.scroll+1)) {
            this.text.text = this.text.text+"\n ";
            //add extra lines to make text divisible by lines displayed
            i++;
        }
        this.text.scroll = 1;
        this.text._visible = true;
        this.calcPageNos();
    }
}
//
//scroll bar methods
//
function setScrollBar(currentPage, totalPages) {
    this.scrollArea.scrollButton._height = (this.scrollArea.scrollAlpha._height/totalPages);
    this.scrollArea.scrollButton._x = 0;
    this.scrollArea.scrollButton._y = (this.scrollArea.scrollButton._height*currentPage)-
this.scrollArea.scrollButton._height;
}
function gotoPercent(percent) {
    p = percent;
    if (p>100) {
        p = 100;
    }
    this.scrollArea.scrollButton._y = (((this.scrollArea.scrollAlpha._height-
this.scrollArea.scrollButton._height)/100)*p);
}

this.scrollUpButton.onRelease = function() {
    _root.textClip.scrollUp();
    scrollUpButton.clicked();
};
this.scrollDownButton.onRelease = function() {
    _root.textClip.scrollDown();
    scrollDownButton.clicked();
};

```

msgBox fla

```

edgeTransform;
centreTransform;
stop();
_root.backIcon.setIcon = setIcon;
_root.frontIcon.setIcon = setIcon;
_root.textClip.textSetup = textSetup;
function setButtons(names) {
    gotoAndStop(names.length);
    for (i=0; i<names.length; i++) {
        edge = new Color(_root["button"+i].buttonEdge);
        centre = new Color(_root["button"+i].buttonCentre);
        edge.setTransform(_root.edgeTransform);
        centre.setTransform(_root.centreTransform);
        _root["button"+i].text.text = names[i];
    }
}

```

```

}
function setIcons(symbol) {
    _root.backIcon.setIcon(symbol);
    _root.frontIcon.setIcon(symbol);
}
function selectButton(button) {
    for (i=0; i<3; i++) {
        if (i == button) {
            _root["button"+i].gotoAndStop(2);
        } else {
            _root["button"+i].gotoAndStop(1);
        }
    }
}
}

//setting colours dynamically --no longer uses array as harder to pass to from Director
function setColours(er, eg, eb, et, cr, cg, cb, ct) {
    _root.edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    _root.centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    bo = new Color(_root.border);
    bt = new Color(_root.backTop);
    bm = new Color(_root.backMiddle);
    bb = new Color(_root.backBottom);
    bi = new Color(_root.backIcon);
    bo.setTransform(_root.edgeTransform);
    bt.setTransform(_root.centreTransform);
    bm.setTransform(_root.centreTransform);
    bb.setTransform(_root.centreTransform);
    bi.setTransform(_root.centreTransform);
}
function textSetup(textToDisplay) {
    this.text._visible = false;
    this.text.wordWrap = true;
    this.text.multiline = true;
    this.text.selectable = false;
    this.text.text = textToDisplay;
    this.text._visible = true;
}
function setIcon(icon) {
    if (icon == "info") {
        this.gotoAndStop(1);
    }
    if (icon == "question") {
        this.gotoAndStop(2);
    }
    if (icon == "error") {
        this.gotoAndStop(3);
    }
    if (icon == "disk") {
        this.gotoAndStop(4);
    }
}
}

```

playlist fla

```

_root.fileWindow.updateFileWindow = updateFileWindow;
_root.fileWindow.scrollBar.setScrollBar = setScrollBar;
_root.fileWindow.scrollBar.gotoPercent = gotoPercent;
//colour settings
function setColours(er, eg, eb, et, cr, cg, cb, ct) {
    _root.edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    _root.centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    fc = new Color(_root.fileWindow.panelBoxCentre);
    fe = new Color(_root.fileWindow.panelBoxEdge);
    fe.setTransform(_root.edgeTransform);
    fc.setTransform(_root.centreTransform);
}
//
function updateFileWindow(files) {
    this.files = files;
    this.select = select;
    for (i=0; i<16; i++) {
        if (this["element"+i] != null) {
            //if a previous item is there hide it
            this["element"+i]._visible = false;
        }
    }
    for (i=0; i<files.length; i++) {
        if (this["element"+i] == null) {
            //if the items are already there dont make new ones
            this.attachMovie("fileElement", "element"+i, this.getNextHighestDepth());
        }
        this["element"+i]._visible = true;
    }
}

```

```

        currentItem = this["element"+i];
        currentItem.setHighlight = setHighlight;
        currentItem.setIcon = setIcon;
        currentItem.setIcon(files[i][0]);
        currentItem.text.text = files[i][1];
        currentItem._x = 40;
        if (i == 0) {
            currentItem._y = 20;
            //currentItem.setHighlight(true);
        } else {
            currentItem._y = (this["element"+(i-1)]._y+this["element"+(i-1)]._height+2);
        }
    }
}
function setIcon(icon) {
    if (icon == "none") {
        this.gotoAndStop(1);
    }
    if (icon == "played") {
        this.gotoAndStop(2);
    }
    if (icon == "error") {
        this.gotoAndStop(3);
    }
    if (icon == "play") {
        this.gotoAndStop(4);
    }
}
//0xFFFFCC
function setHighlight(level) {
    //added to each fileElement
    if (level == 2) {
        this.text.backgroundColor = 0xFFFF00;
        this.text.background = true;
    } else if (level == 1) {
        this.text.backgroundColor = 0xFFCC00;
        this.text.background = true;
    } else {
        this.text.background = false;
    }
}
function select(itemNo, subSelections) {
    //subselections is an array of itmes to highlight dim
    for (var i = 0; i<this.files.length; i++) {
        currentItem = this["element"+i];
        if (i == itemNo) {
            currentItem.setHighlight(2);
        } else if (subSelections.length>0) {
            for (var n = 0; n<subSelections.length; n++) {
                if (subSelections[n] == i) {
                    currentItem.setHighlight(1);
                    break;
                } else {
                    currentItem.setHighlight(0);
                }
            }
        } else {
            currentItem.setHighlight(0);
        }
    }
}
//scroll bar methods
function gotoPercent(percent) {
    p = percent;
    if (p>100) {
        p = 100;
    }
    this.scrollArea.scrollButton._y = (((this.scrollArea.scrollAlpha._height-
this.scrollArea.scrollButton._height)/100)*p);
}
function setScrollBar(currentPage, totalPages) {
    this.scrollArea.scrollButton._height = (this.scrollArea.scrollAlpha._height/totalPages);
    this.scrollArea.scrollButton._x = 0;
    this.scrollArea.scrollButton._y = (this.scrollArea.scrollButton._height*currentPage)-
this.scrollArea.scrollButton._height;
}

```

scrollbar fla

```
_root.scrollBar.setScrollBar = setScrollBar;
```

```
function setScrollBar(currentPage, totalPages) {
    this.scrollArea.scrollButton._height = (this.scrollArea.scrollAlpha._height/totalPages);

```

```

    this.scrollArea.scrollButton._x = 0;
    this.scrollArea.scrollButton._y = (this.scrollArea.scrollButton._height*currentPage)-
this.scrollArea.scrollButton._height;
}

```

library fla

```

stop();
var leftPanel;
var leftPos = 30;
var topPos = 60;
var slideSpeed = 80;
var testCurrentSel;
var edgeTransform;
var centreTransform;
if (_root.testCurrentSel == null) {
    _root.testCurrentSel = 0;
}
//
//replacement functions that make calling from director easier
function setupPanel0(titleTxt, subTitleText, noTxt, pageTxt, itemsArray) {
    //slideToPanel(0);
    panel0.setupPanel(titleTxt, subTitleText, noTxt, pageTxt, itemsArray);
}
function setupPanel1(titleTxt, subTitleText, noTxt, pageTxt, itemsArray) {
    //slideToPanel(1);
    panel1.setupPanel(titleTxt, subTitleText, noTxt, pageTxt, itemsArray, panel0.getGlobalSelPoint());
}
function setupPanel2(titleTxt, subTitleText, noTxt, pageTxt, itemsArray) {
    //slideToPanel(2);
    panel2.setupPanel(titleTxt, subTitleText, noTxt, pageTxt, itemsArray, panel1.getGlobalSelPoint());
}
function setupPanel3(titleTxt, subTitleText, noTxt, pageTxt, itemsArray) {
    //slideToPanel(3);
    panel3.setupPanel(titleTxt, subTitleText, noTxt, pageTxt, itemsArray, panel2.getGlobalSelPoint());
}
//
// colour settings code
function setColours(er, eg, eb, et, cr, cg, cb, ct) {
    _root.edgeTransform = {rb:er, gb:eg, bb:eb, aa:et};
    _root.centreTransform = {rb:cr, gb:cg, bb:cb, aa:ct};
    trace("set ok");
}
//
function hidePanel(panelNo) {
    _root["panel"+panelNo]._visible = false;
}
function hideAll() {
    _root.panel0._visible = false;
    _root.panel1._visible = false;
    _root.panel2._visible = false;
    _root.panel3._visible = false;
}
//two functions are used for this - one to set save colours (set by director)
// - the others sets panel colours at run-time (run by other code)
function setPanelColours() {
    pe = new Color(this.panelBox.panelBoxEdge);
    pc = new Color(this.panelBox.panelBoxCentre);
    pe.setTransform(_root.edgeTransform);
    pc.setTransform(_root.centreTransform);
    trace((_root.edgeTransform));
    trace((_root.centreTransform));
}
//
function goToPanel(p) {
    //sets panel p to the left
    _root.leftPanel = p;
    diff = 0+(_root["panel"+_root.leftPanel]._x-_root.leftPos);
    _root.slide((0-diff));
}
function slideToPanel(p) {
    //slides so panel p is on the left
    _root.leftPanel = p;
    gotoAndPlay(2);
}
function slide(amount) {
    _root.panel0._x = _root.panel0._x+amount;
    _root.panel1._x = _root.panel1._x+amount;
    _root.panel2._x = _root.panel2._x+amount;
    _root.panel3._x = _root.panel3._x+amount;
}
function initPanels(gap) {
    for (i=0; i<4; i++) {

```

```

        if (_root["panel"+i] == null) {
            _root.attachMovie("panel", "panel"+i, _root.getNextHighestDepth());
        }
        currentItem = _root["panel"+i];
        currentItem._visible = false;
        currentItem.setupPanel = setupPanel;
        currentItem.setPanelColours = setPanelColours;
        currentItem.setPanelColours();
        currentItem._y = _root.topPos;
    }
    //panel width (before arrow) 315.35
    panel0._x = 0;
    panel1._x = panel0._x+315.35+gap;
    panel2._x = panel1._x+315.35+gap;
    panel3._x = panel2._x+315.35+gap;
}
//menus will stay at full height if they are not passed a selPoint
function setupPanel(titleTxt, subtitleTxt, footerLeft, footerRight, panItems, selPoint) {
    this.panelItems = panItems;
    this.title.text.text = titleTxt;
    this.footer.left.text = footerLeft;
    this.footer.right.text = footerRight;
    this.select = select;
    this.selectArrow = selectArrow;
    this.getGlobalSelPoint = getGlobalSelPoint;
    for (i=0; i<15; i++) {
        if (this["panelItem"+i] != null) {
            //if a previous item is there hide it
            this["panelItem"+i]._visible = false;
        }
    }
    for (i=0; i<this.panelItems.length; i++) {
        if (this["panelItem"+i] == null) {
            this.attachMovie("panelItem", "panelItem"+i, this.getNextHighestDepth());
        }
        currentItem = this["panelItem"+i];
        currentItem._visible = true;
        currentItem.setHighlight = setHighlight;
        currentItem.setArrow = setArrow;
        currentItem.text.text = this.panelItems[i];
        currentItem._x = this.panelBox._x+8;
        if (i == 0) {
            currentItem._y = this.panelBox._y+10;
            //currentItem.setHighlight(true);
            //currentItem.setArrow(true); //use select/selectArrow now
        } else {
            currentItem._y = (this["panelItem"+(i-1)]._y+this["panelItem"+(i-1)]._height+1);
        }
    }
    if (selPoint != null) {
        point = selPoint;
        //adjusts position and height automatically
        this._y = point.y;
        this.panelBox._height = (((this["panelItem"+0]._height+1)*this.panelItems.length)+12);
        this.footer._y = (this.panelBox._y+this.panelBox._height+2);
        while ((this._y+this._height)>(_root._height)) {
            //this works without animation as flash does not update the display until
            calculation is complete
            //trace("y " + this._y + " height " + this._height + " root height " +
            _root._height);
            this._y--;
        }
    }
    this.footer._y = (this.panelBox._y+this.panelBox._height+2);
    //instead of setHighlight above as calcs sel point
    this.select(0);
    this.selectArrow(0);
    this._visible = true;
}
function select(itemNo) {
    for (var i = 0; i<this.panelItems.length; i++) {
        currentItem = this["panelItem"+i];
        if (i == itemNo) {
            //converts x/y point object passed from panelItem to global point object
            this.globalSelPoint = currentItem.setHighlight(true);
            this.localToGlobal(this.globalSelPoint);
        } else {
            currentItem.setHighlight(false);
        }
    }
}
function selectArrow(itemNo) {
    for (var i = 0; i<this.panelItems.length; i++) {
        currentItem = this["panelItem"+i];
        if (i == itemNo) {

```

```
        currentItem.setArrow(true);
    } else {
        currentItem.setArrow(false);
    }
}
function getGlobalSelPoint() {
    return this.globalSelPoint;
}
//
//functions for panelItems
function setArrow(boolean) {
    if (boolean == false) {
        this.gotoAndStop(1);
    }
    if (boolean == true) {
        this.gotoAndStop(2);
    }
}
function setHighlight(boolean) {
    if (boolean == true) {
        this.text.backgroundColor = 0xFFFF00;
        this.text.background = true;
        var point:Object = {x:this._x, y:this._y};
        return point;
    } else {
        this.text.background = false;
    }
}
```

Activity Log

Date	Log
Monday 11/10/04	<ul style="list-style-type: none"> • Meeting with supervisor. • Discussed ways of approaching project using Java, including using the JavaSound module, Java Media Framework, JSResources web site and Swing GUI. • Started proposal and log. • Research into alternative Media Centre software. • Research into infrared remote interface devices for the PC.
Thursday 14/10/04	<ul style="list-style-type: none"> • http://www.javaworld.com/javaworld/jw-04-1997/jw-04-jmf.html Reading about Java media framework. The framework should provide cross-platform compatibility for audio/video playback.
Monday 18/10/04	<ul style="list-style-type: none"> • Experimented with Java swing demo. Showed possibilities for making items on screen.
Wednesday 20/10/04	<ul style="list-style-type: none"> • Research into alternative media centre software and remote control devices.
Monday 25/10/04	<ul style="list-style-type: none"> • The Java media framework is capable of playing most legacy formats, e.g. MPEG1 (VCD), MP3, WAV. It can do this without needing to access the operating systems codec's. However newer less-established codes (DivX/XViD) may be harder to implement (need operating systems installed codecs). • MDIApp.java example up and running. Example plays multiple media files (opens up multiple players in a window). It also uses swing for the GUI. • Discovered that the Java media framework returns its own GUI, but it is possible to use methods to control a player so that it can fit in cosmetically with any design.
Friday 29/10/04	<ul style="list-style-type: none"> • Installed MySQL and set up test database. • Found JavaCoding MySQL/JAVA tutorial. • Ordered JMF Book: JAVA Media API's
Saturday 31/10/04	<ul style="list-style-type: none"> • Books loaned from library: JAVA Database Programming – Brian Jepson The Complete Guide To JAVA Database Programming – Matthew Siple
Monday 01/11/04	<ul style="list-style-type: none"> • Meeting with supervisor. • Discussed using JDBC drivers so the database can type can be changed (e.g. MySQL to tinySQL) without having to make considerable changes to Java code. • TinySQL has an advantage as it requires no mySQL server, and should be able to create its own database. Its JDBC compatibility should ensure it can be replaced with a MySQL database if it is too limiting or inefficient. • Discussed table/schema layout of media database.
Wednesday 03/11/04	<ul style="list-style-type: none"> • Installed tinySQL driver and connected to the JDBC driver. • Tested examples of tinySQL – currently data is not saved to disk though. • Found that mSQL/tinySQL does not support prepared statements, which can be used to enter lots of data without having to compile the statement more than once. • Callable statements may also be useful for procedures that return outputs, but this too is not supported. • Started exercises from loaned book Java Database Programming (1998). However found that mySQL has been developed a great deal more recently (post 1998). A more recent book is required, such as “Java Database Best Practices” – O'Reilly. Reservation made for book.
Friday 12/11/04	<ul style="list-style-type: none"> • Book “Java Database Best Practices” – O'Reilly collected from library.
Thursday 18/11/04	<ul style="list-style-type: none"> • Book "Java Media APIs: Cross-platform Imaging, Media and Visualization" - Alex Terrazas arrived.
Monday 22/11/04	<ul style="list-style-type: none"> • Discovered JMF may be able to play MPEG4/DIVX (i.e. usage of codecs from the operating system), however the book somewhat contradicts this in another chapter. • Discovered the JMF libraries are particularly complicated, even to perform simple operations (playing, pausing, fast-forwarding etc).
Tuesday 23/11/04	<ul style="list-style-type: none"> • Meeting with supervisor. • Discussed using Macromedia Director to create the music centre. The only problem is Director does not have a built-in database facility. However, there are 3rd party database applications (maybe at cost) and the possibility of using a text-file for a database (with possible performance trade-off). • Reading forums for possible database add-ons for Director (known as Xtras). There is a MySQL addon which is not free but apparently does not expire. • MagicSoft's MyXSQL is free for educational/home use, but does have a popup the first time it is initialised. • The ARCA database runs on a MySQL lite engine, and can be used for educational use, but it too has a message box to remind you it is not registered. This has been chosen as the database for Director, as it has cross platform compatibility and a standalone database viewer. • Decided to code in Lingo as opposed to JavaScript as it is more established than Javascript, thus many of the code examples on the Internet are in Lingo. Javascript has only been introduced into Director in the latest version (MX 2004), and is not completely identical to Javascript.

Wednesday 24/11/04	<ul style="list-style-type: none"> • Research into ID3v1 or ID3v2 reader (it would appear that Macromedia cannot do this without an Xtra addon). • Linking to an SWA cast member will read ID3v2, but requires a hacky work around. • The ID3v1 reader/editor and ID3v2 reader require binary IO, which is unregistered and shows another dialogue box.
Thursday 25/11/04	<ul style="list-style-type: none"> • 3d text importing into Web3D maps (perhaps integrating 3D as a future function).
Sunday 28/11/04	<ul style="list-style-type: none"> • Experimenting with Arca DB, creating a DB and using SQL to add values to the DB. • Basic database up and running. • Two ways of adding db entries, recommended way is using lingo values (?, ?, ?, ?). • MP3Parser requires FileBinary and Data.ls to be loaded as well. ID3v1 and ID3v2 reading working. • Found free FileXtra. Retrieves folders.
Monday 29/11/04	<ul style="list-style-type: none"> • So far FileXtra can be used to select a folder (windows GUI), and list the folders and files in a folder (non-recursively). • Special subroutine written to do this, checks the folder list object that FileXtra returns, if an entry contains an \ at the end a subroutine is run to check the folder. • It is best to return all files in the subfolder in a list instead of doing this during DB creation, as it allows the number of files in the folder/subfolder to be counted. This information can be used to make a progress bar that can show DB creation process as it may be slow.
Tuesday 30/11/04	<ul style="list-style-type: none"> • Recursive subroutine written to read a folder and its subfolders and return all files in a list. Subfolder searching can be switched off. Tested with 13,000 files and works ok. • Written ID3 method, which combines the code to get id3v1 and id3v2 to tags; it returns v2 tags in preference to v1 (if v2 fields are missing the equivalent v1 tag is used if present). • Made MP3 files with different combinations of ID3 v1 and v2 tags to check if the tag reader only replaces tags with v2 versions if they exist. The idea is to make best possible use of the tag(s) in the file. • Combined database, folder contents subroutine and ID3 reader so database is now made automatically. DB creation is quite slow, even though this should not need to be performed too frequently, ways in which to speed it up will be required (such as incremental database update – adding new files without clearing the database and starting over).
Wednesday 01/12/01	<ul style="list-style-type: none"> • Interim report started.
Thursday 02/12/04	<ul style="list-style-type: none"> • Meeting with supervisor. • Discussed improvements to Interim Report and creating a Gantt chart to show timeline of project work to be performed. • Idea on predictive text input, use a visual representation of a numeric keypad. During predictive input, grey-out the buttons that will not function because there are no results that match the criteria. For example if F and I have been typed, if there is no artist or song etc. that follows with the letters G, H, I, T, U or V then the GHI (no. 4) and TUV (no. 8) buttons can be greyed out.
Friday 03/12/04	<ul style="list-style-type: none"> • Gantt chart started.
Saturday 04/12/04	<ul style="list-style-type: none"> • Research into IR modules. • The Keyspan remote does not have enough buttons or numeric keys to be operate a fully functional media centre. • The Keyspan Windows and Mac compatible IR remote module can be used with additional remote controls, by creating a specific REM file (although no documentation is available). • Email written to Keyspan about adding additional remote controls. If this is possible, it is a remote solution that can be used on both the Mac and the PC. (NO REPLY 16/01/05) • Igor-USB IR device would be suitable for any remote using Windows. Compatible with Girder software. Instructions are available to build this but components are not readily available. An alternative is to buy a pre-constructed unit from eBay for around £16. This unit does not have Mac compatibility.
Sunday 05/12/04	<ul style="list-style-type: none"> • Experimentation with buttons in Director, and importing Flash elements into Director. • Downloaded 'Using Flash Components' documentation from Macromedia's site.
Monday 06/12/04	<ul style="list-style-type: none"> • Idea On-screen volume display – may require Xtra. • When implementing database, remember to add code to treat characters such as ö, Í, and ü as o, l and u during searches etc.
Tuesday 07/12/04	<ul style="list-style-type: none"> • Meeting with supervisor. • Discussed interim report, Gantt chart and work to be performed over the Christmas period.
Tuesday 28/12/04	<ul style="list-style-type: none"> • Tested database building code WITHOUT adding entries to the database. It appears the Arca Database is slow when entries are added to it. • This is not too much of a problem as the database will only need to be written occasionally. A system that checks for new files and amends the database, instead of deleting and rebuilding from scratch, will make updates quicker. • The FolderRead method has been changed to create a Property List, which separates the file path and filename, as they are often needed independently. This is saves using a string parsing method to separate.

	<p>A file path and filename are saved with each database entry.</p> <ul style="list-style-type: none"> • ID3 reading changed so that: if no genre found, set genre as “other” if no artist/album, set as (no artist/album) if no track name, set as filename • The directory name may be used for the artist field once string reading has been enabled. • To get the file extension, the FileXtra command <code>fx_FileGetType</code> can be used. • The <code>folderRead</code> method now sorts the property list before it is returned. • The <code>itemDelimiter</code> and <code>.item.count[n]</code> is used to divide up a string to get components such as separate folders and the filename & extension separately. Setting <code>itemDelimiter</code> to <code>\</code> will divide up folders, and will divide up file names.
Wednesday 29/12/04	<ul style="list-style-type: none"> • It would be ideal to save configuration settings into a property list, as additional settings can be added to the array list and saved without changes to the saving code. • The <code>value(string)</code> command will convert string back into a property lists, so they can now be saved as a string, and loaded back into a property list. However this has problems with quotes inside strings, and void values. • <code>PropSave Xtra</code> is free and can be used to read/write property lists to disk, and does not have the limitations of the <code>value</code> command. The only disadvantage is that the files written to the disk are not text files and cannot be edited easily. • <code>PropSave</code> now writes the configuration property list to disk on request and loads it at startup. If the <code>CFG</code> file cannot be found or is corrupt, a defaults are loaded (however not saved automatically). • Integrated the extension reader into folder reading. If the folder method is past a list of filters, only the files in the folder that have the same extension are added to the database. • Found an MD5 Hashing Xtra which will turn files, folders and strings into hash values. This may be useful for turning file paths into hash values. This would allow individual property list files to be saved containing the contents of each folder used in the media library. This can be used to see if the folder has changed, and if it has additional media items can be added to the database.
Monday 03/01/05	<ul style="list-style-type: none"> • <code>FILE_MANAGER</code> cast member created to group all methods used to process files, such as tag reading, filename manipulation. • The folder read method has been changed to store the file path and filename as a whole, as this is generally much more needed. 4 methods have been written to get information from path/name: <code>getExtension</code>, <code>getNameNoExt</code>, <code>getFileName</code>, <code>getContainingFolder</code>. • <code>getNameNoExt</code> improved so it works with filenames that include more than one “.” character. • Reading into Parent and Child scripts as part of OO Lingo programming.
Tuesday 11/01/05	<ul style="list-style-type: none"> • Meeting with supervisor. • Discussed work performed over the Christmas holidays.
Saturday 15/01/05	<ul style="list-style-type: none"> • Continued reading into Parent and Child scripts as part of OO Lingo programming. • Designed a prototype remote control layout, which fits an old VCR remote. Searched for Media Centre remote images to get examples of existing remote layouts. • http://eetd.lbl.gov/Controls/pics2/standby.gif
Sunday 16/01/05	<ul style="list-style-type: none"> • Ordered USB Infrared Remote Module (PC only) from eBay. • Organising tools required to fashion the buttons plate for the custom remote. • Continued reading into Parent and Child scripts as part of OO Lingo programming: Understood an important difference between script types. Scripts defined as “movie scripts” can have their handlers accessed anywhere at any time during the movie, and do not appear to have an <code>on (me)</code> method which is run when an instance is created. This explains problems that were occurring when creating instances (child scripts), as this should not be done with movie scripts. Instead, the script type must be changed to a “parent script”. • Every handler declared in a parent script should include “me” as the first variable passed to it, so that it refers to that particular instance (child). If referring to a handler within a the same parent script, “me.handler” should be written. • Idea: When viewing a list of video files, pressing the INFO button will attempt to read a .txt file with the same filename of the video file. If not found, the system will open any txt files found in the same folder as the video. If multiple text files are found, the directional keys < and > can be used to select each text file. Also when videos are playing, have the option of automatically pausing when you return to the menu or options screen. This of course isn’t necessary for music. • Idea: When browsing media, have a playlist creation mode that adds files to the playlist as a default action (instead of playing just the selection). Also have a feature to place file in a specific place in the playlist on-the-fly, instead of placing it at the end, and moving it up manually. • Started reading “Flash – Using Components” PDF and the Flash section of “Director MX and Lingo – Training from the Source” book. It would appear that many of the visual components are best created in Flash. Director has only very basic visual components – even the more complex buttons and input components are Flash Components (SWC).
Monday 17/01/05	<ul style="list-style-type: none"> • Flash MX Bible borrowed from library. Reading “Working with Director” chapter. This shows how a Flash object can send messages to Director. However since the project will be controlled using a remote,

	<p>via keyboard commands, it will be more sensible for Director to receive all input and control the flash object accordingly. If mouse support were added (possibly at a later date) commands from the Flash objects will need to be received by Director.</p> <ul style="list-style-type: none"> To prevent crashing when changing to a frame that the flash movie is not in, make the flash object display in all frames but change its visibility to false.
Tuesday 18/01/05	<ul style="list-style-type: none"> Idea: Add option to play flash movies (those that do not require user intervention). Flash-director integration research continued. Graphical components in Flash appear to only have only one text size that is not large enough for this application. Custom graphical components will be made. To implement the graphical components, child instances of parent scripts should be used. These will manage the menu contents (names that are displayed) and operation, and control flash objects accordingly. For example an array will be passed to the lingo child object, and this will pass the names to the flash graphical component. If the menu is navigated (e.g. up and down) the script will instruct the flash to change the item that is highlighted. However the performance must be checked for items displaying database results to ensure these are displayed fast enough. Invalidate() redraws components in flash (p947) Meeting with supervisor. Discussed Flash-Director integration. USB Infrared remote receiver delivered. Successfully installed and set-up receiver with 'Girder' which receives remote control commands and replicates them with keystrokes.
Wednesday 19/01/05	<ul style="list-style-type: none"> Problem noticed with detection of remote commands: IR device detects buttons on some remotes as the buttons on other. Changing driver/plugin versions did not help. Problem will be investigated further but does not affect work in Director. Email written to Paul Newbury to arrange meeting to discuss best practices for Flash & Director integration. Idea: If music such as REM is searched for, but the music in the database is stored as R.E.M. or R E M, return the results too (and if other punctuation is missed too). Also show album art too (any images in folder). If multiple images found, pick largest (use either dimensions or file size) or one that contains 'front' in filename.
Thursday 20/01/05	<ul style="list-style-type: none"> Further testing of IR module using test program to examine output using the virtual oscilloscope. Contacted designer of IgorPlug IR module explaining problems with the device.
Saturday 22/01/05	<ul style="list-style-type: none"> Research into WMA tag reading. It appears there is no code or xtras available to read tags. It may be possible through the "Windows Media" Media Element but this has not been tested. Research into JPEG information retrieval (such as author, keywords, comments etc.). This too does not appear possible using Director, nor is there an Xtra which enables this. However, this is only a minor function.
Monday 24/01/05	<ul style="list-style-type: none"> Response from Paul Newbury: he advised I contact a Mo Hamdhaidari, a researcher, for advice on Flash components.
Tuesday 25/01/05	<ul style="list-style-type: none"> Emailed Mo for advice on Flash components.
Wednesday 26/01/05	<ul style="list-style-type: none"> Reply from Mo. Meeting arranged for today to discuss Flash and Director. Discussed ways of implementing graphical components (menus etc.) using Flash or native Director components.
Thursday 27/01/05	<ul style="list-style-type: none"> Started first graphical component in Flash: a text file display component. Wrote ActionScript to format text box, calculate number of pages, and operate scroll up/down, page up/down. Discovered how to call ActionScript methods in Flash movies from Director. This is important as it enables easy manipulation of Flash objects, without having to use workarounds such as moving the playback head (callFrame()). Discovered how to send script statements with arguments from Flash movies to Director. This uses the GetURL command, but uses escape characters so that Lingo code can be sent (and not just basic strings). Discovered Flash variables can be accessed directly with the new version of Director, so getVariable() is not needed. This completes communication with Director and Flash using standard SWF movies, so learning to design SWC Flash components is not necessary. Direct-to-stage is the fastest mode for Flash movies in Director, but it also allows them to play at their correct speed (thus ignoring the Director score/FPS). This will be useful if Flash videos are implemented (i.e. not the graphical components) as transparency settings will not be required.
Friday 28/01/05	<ul style="list-style-type: none"> When creating an array to pass to a Flash movie function the sprite(x).newObject("Array",.....) must be used to create it. Attempted to fix problem with calculating page numbers: appears to be a bug in Flash. Problem only occurred with a specific set of text. It was decided that the Flash graphical elements should support the mouse as it is easier to implement this now instead of later. Designed special Flash buttons that use the getURL method to run Lingo code. Clicking a button in Flash attempts to run a Lingo method with the same name as the button's instance. Most buttons in Flash will not be set up to communicate with flash, instead they send a message directly to Director. It is the job of Director to complete the circle and call the appropriate method in the Flash movie. This is because these methods can also be called by keystrokes sent to director (i.e. from the remote control).
Sunday	<ul style="list-style-type: none"> Investigating ways of implementing scroll bars. Although Flash has components such as scroll bars that are

30/01/05	<p>suitable for integrating with the project, they are of a fixed size which is too small. Experimenting with editing the visual elements of pre-made components (by importing the default Halo style format). This revealed more tricky than realised as changing the components size (rather than just their appearance) caused the components not to line up correctly. A decision must be made whether to attempt to fix this, or implement a scroll bar from scratch.</p> <ul style="list-style-type: none"> • Implementing a scroll bar from scratch would require resizing elements based on values about the current page/line and number of pages. Although it should be possible to create a scroll bar that works visibly, implementing one that can be dragged may be more difficult. • Investigation into an entity relationship diagram for the media database. It is apparent that enforcing some cardinality constraints may be a problem as it is not possible to rely on all data being available and valid. For example, ID3 information in MP3 files is often incomplete or inaccurate. Either the missing, but required information can be fabricated, or the cardinality constraints can be relaxed. For example, a track number may not be suitable for a primary key for the track, as you cannot guarantee there will not be two tracks with the same artist/album that have the same number (e.g. if the album is made up of two or more CD's and they have been copied into the same folder).
Saturday 05/02/05	<ul style="list-style-type: none"> • Started work on 4-colour pop-up for menu for fast-text keys. This involves being able to create instances of objects in the library and position them on the stage accordingly. An array will be passed into flash containing the contents of the array. Flash will then create x number of objects (the size of the array) for the menu and name them according to each name in the array.
Monday 07/02/05	<ul style="list-style-type: none"> • Flash action script problems: When creating instances of a movie clip using a string + n to give each one a different ID, referencing each movie clip by using name+n.doSomething() does not work. Using "name"+n or ("name"+n) does not compile, no speech marks (name+n) does compile but does not work. The correct method is using the eval() method: eval("name"+n).doSomething() as this allows strings to be used to reference objects in Flash. The Flash book says that method is depreciated and says _root["name"+n].do is acceptable but it must have _root at the front of it to work. It would appear that these could be used to run methods inside movie clips (e.g. setName, selected) if a button was made to run this code (e.g. _root["inst"+n].select()) but problems occurred when methods of movie clips were called inside the desired for loop. The only way to set properties was to use the depreciated setProperty(item,prop,val) but this couldn't be used to call methods inside, so was not fully suitable. • The book showed how to make a dynamic menu using ActionScript very similar to what needs to be achieved. However this did not show how to call the methods of each movie clip duplicated, and only showed how to change the text on a button by having button.text = newText; in each button movie clip, and changing the variable newText from the main timeline. The example also used currentItem and prevItem variables to point to the menu items derived from _root["item"+n] even though it doesn't seem necessary (probably just to tidy code up). This example also used duplicateMovieClip which required having an instance of a menu item on the stage and hiding the original (not ideal). • New example which uses attachMovie() to create instances directly from the library. • Tip: use this.startDrag to allow people to move the scroll bar in the text reader window. The number of pixels the mouse has moved can be used to detect where the scroll bar should snap to.
Tuesday 08/02/05	<ul style="list-style-type: none"> • Experimentation with createMenu200 from the Flash CD. This example successfully uses attachMovie() but still the code cannot be modified to run methods inside each button (menu item). Instead the background colour of each item can be modified from the code in the main timeline. Although this may not provide the best object-orientated design it will be used to prevent any more time from being spent on the design. • Created a popup menu that will open and close smoothly. It would appear that by changing a movie clips _height by + or - an integer it sometimes sets itself to a value that is not an integer, even if int() is used. When decrementing height by 1 • Experimentation with changing the colours of shapes on the stage using ActionScript (so coloured buttons can be made on the fly). • Meeting with supervisor. • Discussed work and extension if necessary. • Contacted Kerith Harris. • Found a solution for the methods problem! If a method is put inside a button it doesn't seem to work when you call it, however if the function is assigned to it using movieClip.functionName = this Function; then it works. All that must be done is the functions defined in the main timeline and assigned to every object that needs them.
Wednesday 09/02/05	<ul style="list-style-type: none"> • It would appear that _yscale is a percentage of an objects original height. However if you add extra to _height, _yscale will then be >100% as it always reflects its original height on the stage. • To make instances of movie clips using attachMovie, the clips properties "export for ActionScript" and "export in first frame" must be enabled.
Thursday 10/02/05	<ul style="list-style-type: none"> • Problem discovered with the placement of objects in a movie clip when compared to objects on the stage. The _x and _y are at zero on the stage when an object is in the top left corner, but this refers to the centre of the movie clip they are in (denoted the + symbol). If objects are above the + symbol they have a - value. Instead of moving objects, coding will be rewritten so no static values will be used for stop points. Instead, references to the _x and _y coordinates of objects on the stage will be used.

	<ul style="list-style-type: none"> • In addition, the point on which the _x and _y values are measured on the movie clip you are moving is also represented by the +. So if graphics are centred in their movie clip the _x and _y values are taken from the middle. • When making the fasttext buttons flash yellow, it was easier to create an extra layer with a yellow box on it rather than change the colour using the = new Color system. This is because when the colours are set initially the Color objects are nullified, and getRGB only works after it has been set initially (i.e. cannot retrieve colour information). • Fast text buttons complete. Although this took some time, fundamental Flash knowledge was learnt that will make creating the other graphics far easier.
Friday 11/02/05	<ul style="list-style-type: none"> • Main menu design developed. Main menu is to allow each item to have a sub-menu if desired. Main menu will also be a dynamic menu (menu items created with code). This will allow menu to be edited easily, menu items to be enabled/disabled and user-menus to be created. The menu will only have a submenu of one level deep. • The setRGB methods to set-up button colours have been replaced with setTransform. setTransform is better as it does not change the object, it is more like a filter and can be set to 0. A setRGB object can be changed to another colour but original colours cannot be restored. Buttons are now made black, and the transform adds RGB offsets to change the colour. This system also allows gradients to be added to add depth to the buttons. • When previewing flash movies in Flash it is important to ensure “show all” is ticked in magnification otherwise objects that are supposed to be centred may not show properly. • During a menu transition several things happen: the menu transparency is increased, the h_scale and w_scale reduced, and the _x and _y reduced. To keep these all in sync (they all have different maximum and minimum values that must be adhered to) a percentage variable transAmount is used. This is changed from 0 to 100. For each value to be changed (width, scale, opacity etc) a mathematical formula has been made for each to calculate a value between its max and min values. This system would also be preferable in the animated fast text menus as it is more simple/reliable since there are no concerns with property values wandering off. E.g. when setting some variables (height etc) even though they are set to an integer they sometimes have a decimal value. • There is a problem with fonts in the menu transitions. When the alpha value is changed with ActionScript, it is not applied to the font. According to web sites this is a problem with the font not being embedded. This has been attempted but is still not working correctly.
Saturday 12/02/05	<ul style="list-style-type: none"> • A part of the text properties dialog box was hidden. The character button allows you to select characters to embed (best to do only the ones you need). After this the text is dimmed successfully. • Converted previous Flash clock project to a minimalist clock for the front display.
Sunday 13/02/05	<ul style="list-style-type: none"> • Update text window to allow communication with lingo and scroll bar. The scroll bar allows you to scroll up/down, plus page up/down (by clicking below and above the slider). The bar however does not allow dragging. Although this can be enabled, the bar is really a visual aid for when the centre is used via remote control. Dragging will not be possible on any objects using the remote.
Monday 14/02/05	<ul style="list-style-type: none"> • Constructed a progress bar that accepts a value as a %. • Started creating display components for the configuration (checkboxes, radio buttons, text input). Selected symbols and constructed graphics for other icons within the media centre. • Created a graphical file browser. Browsing is not performed using a tree, however a mini-tree display is shown that shows the current folder and the two higher folders in the tree (if any). Each item in a folder contains an icon to show its type (picture, video, audio, text) and the icon colour corresponds to the menu colours. • Started the structure for the draft final report. • ADD lingo click to file elements.
Tuesday 15/02/05	<ul style="list-style-type: none"> • Final report contents (structure) constructed. • Meeting with supervisor. • Discussed final work performed in Flash, and report structure. • File elements made selectable. • Configuration display components merged into a single configurable component. • Solved the problem with functions inside movie clips that do not run if they have been called from code attached to the stage. Use this.onEnterFrame = function(){} to call methods inside movie clips. There are problems with this though... • IMPORTANT: Must remember to test, and if necessary, remove \ from filename strings sent to Flash as flash may interpret them as some sort of escape character. This is not just a problem when the backslash occurs at the end of the string, but also in the middle of the string. The character following it is missing.
Thursday 17/02/05	<ul style="list-style-type: none"> • Problems discovered with onEnterFrame method, and when code for an object is placed inside it, instead of being assigned to it. The problem is: if an instance of an object is made, and if in the same method you try to call a function inside the object it won't work. The function will only work if it is called in another method (such as in a separate). Basically, if you create a new instance of an object then try to run the method straight after it will not work. However if you assign the new instance the method, you can run it in the next line of code if required. • Feature: tracks view of music library should highlight tracks if they are playing, or in the active playlist.

	<ul style="list-style-type: none"> Designed graphics for the audio database browser. A 2-up grid was designed (two columns at a time) so long text can be displayed.
Friday 18/02/05	<ul style="list-style-type: none"> Completed predictive window. This will pop up whenever a search is performed, and will offer a list of words that match the inputted text. Set-up a scrolling system for the media library browser. This will scroll from left to right when a third column is viewed. The menu can also be set so that when progressing through the menu, each submenu will resize itself to the size of its contents, and align its top to the selection on the previous menu. If the submenu is to hang off the bottom of the screen, it is moved up until the full menu is displayed.
Tuesday 22/02/05	<ul style="list-style-type: none"> Meeting with supervisor. Discussed the report outline.
Friday 25/02/05	<ul style="list-style-type: none"> 800x600 is the chosen resolution for Director. This is because it is the nearest computer resolution that is greater than PAL 720x576. Planning/initial work on Final Report. Adding graphical Flash components to Director. Remote interface (though keyboard) set-up with Director. It would appear that some (most non-alphanumeric keys) of Girder's keyboard emulation controls Director instead of the program. This will only be a problem while the software is developed in director. To solve the problem short-term, the remote will emulate alphanumeric/punctuation keys only and Director will be set-up to listen to these keys. Girder's "Alternative Proc." Did not help to resolve the situation. Problem discovered with keydown and keyup. Keydown does not work when the movie is not playing (e.g. after an on frame hold command). It can be replaced with keyUp which does, but keyup returns none or conflicting (incorrect truncated) values for keyCode, keyPressed. However since alphanumeric keys are being used key works, but may not work with system keys.
Saturday 26/02/05	<ul style="list-style-type: none"> The keyUp command was initially made a movie script, which attempted to run methods in the current frame. However if the current frame did not contain the handler that was called (e.g. for a particular key or a general handler) the movie would crash. Now, the on keyUp is put into each frame which requires input. It calls a method that returns the name of the key pressed. This is so that the key names are standardised and logical throughout the program, but they key binds can be changed or the method in which the keys are detected can be changed. The frame script then calls its own method for the key name returned, or whatever appropriate. Discovered how to make arrays within arrays for Flash in Director. Created a selection of audio files of popular formats to test Director's playback capability.
Sunday 27/02/05	<ul style="list-style-type: none"> Experimentation with Windows, Real, Quicktime media elements. All sometimes automatically resize to video, or stretch the video to the window which causes problems with widescreen/letterbox video. The solution is to read the videos height and width parameters and change the video window sprite position and dimensions accordingly. Note: if you divide an integer by another the result will be a an integer too - must add 0.0 every time. As most video types in Director are direct-to-stage (as they are overlays) an OSD is harder to implement. Instead the video will be 'squashed' horizontally so an OSD can be displayed for a short period of time. OSB built in flash that displays name of program, a play/pause/ff symbol, time and a progress bar.
Monday 28/02/05	<ul style="list-style-type: none"> Work on report for supervisor meeting.
Tuesday 01/03/05	<ul style="list-style-type: none"> Meeting with supervisor. Discussed report, particularly the MVC paradigm. Feature: change video/photo background colour. Also if an album contains songs with different genres, list the album in its entirety under both genres.
Tuesday 22/03/05	<ul style="list-style-type: none"> A big difference between Director and Flash is that Directors on enterFrame and exitFrame run repeatedly if you want it to stay in the same frame. To get round this, an on beginSprite and endSprite is used which works like Flash, only running once. The 'do' method in Director is equivalent to eval() in Flash.
Wednesday 23/03/05	<ul style="list-style-type: none"> It is somewhat difficult to implement a dynamic menu due to the way arrays are passed to Flash newObject("Array",x,x,x) etc. Code is required that can generate a statement to send to flash which will depend on the number of menu items in the array. For now the menu will be fixed. However a solution for the director-flash arrays problem has been made for the fasttext menu. A director array is read and the items are converted to a string with commas between each. The rest of the code is added in the form of stings then run using the 'do' command. One drawback is these can only be checked at run-time. Remember to make the menu items 'flash' if necessary.
Thursday 24/03/05	<ul style="list-style-type: none"> Redesigned coding for fasttext keys. Each frame is responsible for setting up the fast text menus, and opening/closing them when the coloured keys are pressed. Once a menu has been opened, the up/down/enter/back keys are redirected directly to the fasttext code, and other features on the frame cannot be navigated until a fasttext selection is made or the menus are closed. The frame is still responsible for passing key presses of the coloured buttons, to allow changing of menus or closing the current menu. An important consideration is each frame must not redirect the coloured key-presses for menus it has not set-up.

	<ul style="list-style-type: none"> • Integrated flash clock into main menu. Colour is set from director (same as the rim of main menu buttons).
Friday 25/03/05	<ul style="list-style-type: none"> • Research into database design by looking at XBMC/Meedio database.
Saturday 26/03/05	<ul style="list-style-type: none"> • Designed a suitable database schema and implemented it in SQL. Written code to populate database correctly. Change folder reading back so folder reading returns path, and file name separately. Folder reading no longer uses a property list, it uses a normal list. However the sort function works successfully.
Sunday 27/03/05	<ul style="list-style-type: none"> • For predictive text searches, since the list of results is refined each time a letter is pressed, it will probably be better to produce a temporary table for predictive results, and refine this to another temporary table each time a letter is pressed. However in testing the function is so quick it may not be necessary. Also handling of temporary tables will be required. • The createSelection is an alternative to executeSQL in Arca. It allows smaller groups of results (or just the number of rows to be returned) without having to return loads of information to a slow property list that is no needed. • For the input numbers for predictive text, it has been decided to keep these as strings, as the integer() method does not reliably convert large numbers for the same reason. This is because numbers greater than 10 are treated as a power. • There may be a bug in the Arca Database Xtra. ? can be used in the SQL code, and the variable in its place is passed as an array afterwards. However, if a ? is used to specify the field name to return, it returns x amount of the field name instead of the actual field names. To get round this, the entire SQL string including the front bit (SELECT ETC FROM ETC) is generated and passed to Arca as one whole string. • Winamp does not allow searches such as royksopp, but this does.
Monday 28/03/05	<ul style="list-style-type: none"> • Predictive results tested in predictive window. • Problem with Director. It will freeze if a string very large in size is passed to Flash (such as searching by song and pressing 1 button only). Results should be lowered to prevent crashing. • When results were properly formatted there were no problems with Flash crashing. It may be to do with one long line of text is passed and not the amount. • Improved behaviour of the predictive window so that numbers are not typed, but the letters of the first result are. • Writing M3U reader writer. From examining a handful of M3U files from different sources it was noticeable that although there is a standard, they vary. Most contain #EXTM3U at the top to denote they are M3U files, and each entry comprises of two lines, the first #EXTINF and the file length (seconds), and the name to be displayed (usually from the ID3 tags), and the second line is the path. Some files just contain the path on each line. • It was also noticeable that the return character can vary. Some files when opened in Notepad appeared on one line but had a □ where there should normally be a hard return. These also caused problems with detecting the hard return in Director when setting the itemDelimiter. To get round this the type of line break was detected by testing for RETURN and stringToChar(10) which represents the square. • http://cboard.cprogramming.com/archive/index.php/t-31212.html - mod tip
Tuesday 29/03/05	<ul style="list-style-type: none"> • Updated ID3 tag reader so any track numbers of the xx/xx format are converted to xx. • ID3 tag reader also automatically converts 'The Band' to 'Band, The' • Writing method to detect txt files about the content being viewed. The method should also produce file info if possible. • Writing method to detect Album Art in folder. Attempts to detect front cover art and categorise images based on keywords. • Improved db code – less repeated code. • New idea for predictive dictionary. Written initial code to generate dictionary and initial tests performed. This will be looked into as an extra. It is slightly more complex.
Wednesday 30/03/05	<ul style="list-style-type: none"> • DB update script written. Problem with case sensitivity with path names detected. Should not normally be a problem as the paths will be stored in the program and will not change. • Visual file-browser for browsing OS, CD's, Memory cards etc.
Thursday 31/03/05	<ul style="list-style-type: none"> • Continued work on file browser.
Friday 01/04/05	<ul style="list-style-type: none"> • Unlike Flash there is no ceil maths function in Director. However a simple one can be made by: integer(no + 0.4999999) • Most media centres do not show a tree, which is useful so you understand the structure. This one creates a mini-tree of the last 3 folder levels. • Started text reader and other windows that will be needed occasionally on multiple frames of the movie. Research into creating instances automatically using lingo, but some say this can be unreliable unless a dummy is made.
Saturday 02/04/05	<ul style="list-style-type: none"> • It is apparently unreliable to attempt to create flash instances of flash sprites on the stage, so it is still best for boxes such as the dialog box and text reader window to be created hidden. • Problem with hidden flash sprites. When they are hidden or completely off the screen they are not rendered and therefore not accessible (no flash engine loaded). However making one from invisible to visible does not work as it does not have any time to initialise. Tried changing the idle load period but it did not help. Also tried displaying the sprite and then closing it. But of course, if it was not drawn initially the making it

	<p>visible then invisible does not occur since the flash sprite will not attempt to draw until there is no code running.</p> <ul style="list-style-type: none"> • One solution was to put a getURL command in the Flash that called a lingo script. This ensured those functions were only called when the Flash object had initialised. However the problem was that if it is initialised, when making it visible because the Flash object is not initialised for the first time the code is not called. Solutions could be made for this but the final solution is... • Two scripts before the main part of the movie called preStart and start are used to display the flash sprite and then hide it. Because these are on different frames it ensures the sprite is fully initialised before moving onto the next frame (they cannot be called on the same frame). This does mean the flash objects 'flash up' for a second but this is hidden by a black rectangle! This seems the most reliable solution, and indeed they can now be hidden without a problem (it is only if they are hidden from the start they don't initialise) but this ensures they initialise once per run. • Made a copy of text reader window as a dialog box. Dynamically configured to select title, text, background icon, number of buttons. Instances controlled in same way as above. Mini script written to show a dialog box when there is an error instead of alerts and puts.
Sunday 03/04/05	<ul style="list-style-type: none"> • Further experimentation with audio playback. Using Directors sound() object will provide best compatibility, but it is not capable of playing all types of media. A priority system should be introduced where depending on the type of media is should first be played using the sound object, and if not an appropriate player (such as WMP) is found. • Experimentation with Director's text boxes. Unlike flash it would appear that you can only set the text of the cast member, and the sprite instances on the stage cannot be set individually. I find this behaviour of director annoying. This means that if you wanted 6 identical text boxes on the screen but with different text in each you need a cast member for each text box. As all instances of the same text box must share the same text. Flash label components can be used but these effectively introduce more flash onto the stage, and require special styles to apply formatting, which is too complicated for such a simple label. • Windows media and real media plug-ins will play without a sprite instance needed. However QuickTime must have a sprite on the stage, so if the user navigates away from now playing the sprite must still be displayed. • Method written that determines the type of media and plays it in the suitable player. It is possible however that the internal sound player does not allow seeking. However, a compatibility mode could be enabled that disables seek but allows playback. • Problem with Windows Media. When controlling the cast member with play() and stop(), every time the audio restarts another instance of the media is played (slightly out of sync causing an echoing effect).
Monday 04/04/05	<ul style="list-style-type: none"> • To solve the problem with multiple files being played, sprites are made of each cast member, and the play/stop commands are sent to them. At the start of the movie each one is sent a stop command and a dummy wav file to ensure no sound is played (they like to play as soon as they appear in the frame). The boxes are invisible but have been put off the stage anyway. • Quicktime time works in ticks so a ticks to milliseconds converter is required. A tick is 1/60th of a second, a millisecond is 1/1000th of a second. • Due to the different players there will always be some inconsistencies. E.g. the wma player rew() command will not rewind by 10 sec unless the song has played for at least 10 seconds. • A way must be made to differentiate between real video and audio files. Real files carry extensions 'ram', 'rm' or 'ra' but there is no apparent standard followed as to which denotes what type of file. Of course detection will not be possible when browsing (probably too slow) but just before playback so the appropriate mode is selected. • Video playback works the same as audio, whereby the appropriate player is chosen for the file. However even though the video overlay is in front of everything else, Flash sprites appear to have a detrimental effect on the performance. These should be hidden when the video is in full display. • Added DVD-specific buttons to remote control fasttext keys.
Tuesday 05/04/05	<ul style="list-style-type: none"> • Checking new database with a large amount of files: building, adding more media, removing media, changing some media. • The fasttext menu code has been changed so that it can be shown and hidden without any problems (like with the message boxes). • When the video is playing it has been decided that the FT menus should not open but only display their function, otherwise the menu will not be seen due to it being behind the video overlay. • There is a problem with Quicktime when in direct-to-stage mode and flash sprites. Although the flash sprite is hidden it causes the video to flicker. This is a problem with the FTMenuu which overlaps. Since the menus are not actually required, it will be removed and replacement buttons will be placed as part of the OSD menu. • Video playback script now universal and controls all players. • Attempted DVD playback but adding extra fails and returns an error code. Others are experiencing the error on the Internet but no-one yet seems to have found a solution.
Wednesday 06/04/05	<ul style="list-style-type: none"> • Designed playlist in Flash derived from file browser window. An icon appears next to the track if it is playing, played, not available and highlights in light yellow if is part of a multiple selection.
Thursday	<ul style="list-style-type: none"> • Writing play list manipulation code. Playlists will be in a standard format and can be passed round the

07/04/05	<p>program (such as for loading/saving). Video playlists will be made too (called video queues).</p> <ul style="list-style-type: none"> • Need to make ceil a global method.
Friday 08/04/05	<ul style="list-style-type: none"> • Bug discovered in Flash/Director integration. It would seem that when a flash array is made using the newObject("Array",) command if a single number is passed it does not work and creates an array of type undefined. However you can send a single string, that works. The flash code for creating selections seemed to work but not from director. To solve this, for this bit of code an extra number is put in the array greater than 10. Since the only purpose of the array in this case is to check a number of 1-10 for an entry the same in the array, a number greater than 10 will have no effect. • Playlist editor complete. • Started writing code for library. The createSelection method built into the Arca database will be used to manage the selections made by users.
Saturday 09/04/05	<ul style="list-style-type: none"> • Constructing database code to query the database. The library panels are to list alphabetically artists, genres, tracks etc. When a user searches predicatively the table should spring to the matching result. However when using Arca's create selection, we do not know which page of rows contains the matching result as we cannot query an Arca selection. Also the primary key (e.g. Artist ID) may not be a reliable method when artists are added and a sort is done. So a temporary table is made when the library is initialised that is effectively a duplicate of, say, the Artist table, but with an extra column that has an ID. When a user searches and the predictive method detects a match, a further SQL statement will take the match and find the RowID for the result. This will then be used in conjunction with the getRows method of Arca. We now know the row number that our result is, all that is needed is to return the correct page of results that contains the row. Note that results are always returned in pages of (e.g. 10). • To achieve this when using INSERT INTO and using a SELECT statement at the end (for multiple rows), if the SELECT statement contains ORDER BY it is ignored. However, if a CREATE VIEW is defined it will ensure a duplicate table will be made that contains the results in order. The benefit of this is quite simply the row numbers in the createSelection will reflect our temporary table, thus we can examine what row we are to find a result. • One annoying Director bug is single line if statements, if they appear above an end if of a higher if statement, director thinks they belong to the single-line statement (and of course director thinks code is missing). This can be resolved by putting – below them to separate – even though this shouldn't really make a difference as the compiler should ignore it. • Need to update predictive code to use the temporary tables created as the tables get refined. It can now be used with both, plus returns ID numbers so songs/artists etc can be found quickly in the database.
Sunday 10/04/05	<ul style="list-style-type: none"> • Continuing library code. It has been decided to keep the RIGHT> button and SELECT button to mostly do the same, to prevent confusion, even though the library would benefit from having these two objects separate. • It has been decided that adding an "all option", although very useful, is very important. Ideally it would only show as item 1, so if 10 pages were to be displayed, the first page would show the ALL item + 9 others. The subsequent pages would contain 10 items. It is not as simple as adding an extra 1 as the number of pages because of the maths used to calculate the rows returned. I.e. a statement would be required that ensures on the first page only the first 9 items are retrieved, however that will upset the counting. • A solution has been decided that either (a) the all option is to appear at the top of each page of results (less desirable), or (b) the first page of results contains an extra result ALL. This does mean that a line is wasted on all subsequent pages but is far more simplistic. The view could also be disabled if necessary. • Like other applications, the program crashed due to the " marks in an album name. This is one particular disadvantage of using the do command as the string is taken literally, and any escapism handled by Director is lost. Unfortunately there may not be a way round using the do command, but escape characters should be investigated at the database level. The poor handling of quotes is a big problem and affects the first page of albums, and tracks. Although a problem like this has been seen in other MP3 applications it must be fixed. The library has a method to convert arrays to comma delimited strings for use when passing to flash. This now uses a procedure found on a Mediamacros site to replace all " with ` . This procedure should be made global for use with all methods that pass arrays to Flash.
Friday 22/04/05	<ul style="list-style-type: none"> • Continued re-organising code. • Research into Director's exception handling. • Message system devised for when items are selected from global flash menus such as the message box and fmenu. • Complete playlist manager. • Started code loop for playlist running.
Saturday 23/04/05	<ul style="list-style-type: none"> • Completed playlist code.
Sunday 24/04/05	<ul style="list-style-type: none"> • Started info reader for files. A special MP3 file MAXINFO was made with complete ID3 data. • When new images are imported into director they change size (as with the video) so code is needed to ensure they are repositioned every time. Initially it was thought that it would be better to use a Flash object, but there were also problems with that (image importing at run-time was restricted to non-progressive JPEGs and importing the JPEG to the root effectively cleared the flash movie of everything else, including

	<p>the code to set the bitmap!</p> <ul style="list-style-type: none"> • To get Director's image working properly then the registration point must be moved to the top left. Once performed, it appears to stay. Currently images are stretched to fit the square which is desirable for most images. • Got albumart working but does not slideshow through images. This can be added when the slideshow is made.
Monday 25/04/05	<ul style="list-style-type: none"> • Picture browser working. Unfortunately when loading the thumbnails director effectively loads each file inside itself which makes browsing of large photos painfully slow. • The file browser, slide show and picture browser all share the same types of array objects so they can be passed seamlessly from item to item. • Created a self-maintaining backHistory array that records the browsing throughout the media centre so the back button can be used appropriately. This is initialised on startup with all the labels. The property list given by Director is converted into a standard array.
Thursday 28/04/05	<ul style="list-style-type: none"> • Wrote code and linked browsers so items can be queued up, played.
Friday 29/04/05	<ul style="list-style-type: none"> • ADPCM files are not supported by Director. • Upon WMA audio playback, the movie stage appears to loose focus although focus does not appear to be taken by anything else. A mouse click on the stage is required to make the system respond to remote commands again. • The biggest problem with Director is everything has to be on different sprite channels as the visible command makes the sprite channel invisible.
Saturday 30/04/05	<ul style="list-style-type: none"> • Completing the library predictive text code.
Sunday 01/05/05	<ul style="list-style-type: none"> • Enabling queue-up from the library.
Monday 02/05/05	<ul style="list-style-type: none"> • Final touches before demonstration and organisation of sample media. • After playing Video files, flash objects are not always initialised in time. Two special frames (similar to what is used at the start of the movie) are used to ensure they are initialised.