Avid4 Software Design

# Introduction

Avid4 runs on a Windows Media PC and plays many different forms of audio and visual content using a variety of players, both internal to the media PC and external. Output is via an HDMI-connected screen and a multi-zone AV receiver.

The component hardware and player software applications, together with necessary audio/visual hardware, are all controlled and coordinated by the Avid4 software, which runs under Windows 7 or Windows 8. The source for this is organized as a single Visual Studio solution constructing three separate modules.

This document describes the architecture and design of that bespoke software. Note that all the Avid software described here is freely available as open source under the MIT License.

The description in this document is at a relatively high level, showing how each part fits in the context of the complete system and how each relates to other parts. Detailed descriptions as to how each part works can be obtained from the source code which is commented.

# Software Modules

The primary Avid4 software module is a web application **Avid4G.Net**, which runs within the context of the Windows IIS web and application server. This presents the Avid UI to any connected touch-capable browser (e.g. phone or tablet), accessing all the devices and players via its “middleware” classes.

However, a web application within IIS has no ability to access the logged-in desktop in which the player applications execute. Avid4G.Net necessarily runs as an isolated “worker” web application process with no desktop capabilities. So, to allow desktop access, two additional components run as system tray applications within the desktop session of the logged in user. These components are then accessed by the **Avid4G.Net** web application through networked inter-process communication.

These two desktop components are:

* **Avid.Desktop**, responsible for launching and controlling player applications, together with other miscellaneous simulated user input functionality.
  + **I**n addition, this component is responsible for discovering the address and service URLs for the Sky set-top box web services on the local LAN; this responsibility is bundled here simply for convenience.
* **Avid.Spotify**, responsible for selecting and playing music streamed from Spotify.

In principle, only one desktop application is necessary for both these responsibilities. But they have been kept separate to allow the Avid.Spotify component to be used on its own outside of the overall Avid4 structure in other environments.

In addition there is a publicly hosted web application **Avid4SpotifyAuth**, which runs somewhere on the public internet to provide the necessary authentication handshake and to hold the necessary secret key to confirm the identity of the accessing software.

All software modules use managed code with the .Net 4.5 execution environment.

# Avid.Desktop

Avid.Desktop runs with administrator rights in the logged-in user’s desktop session. It has no user interface other than a single icon in the system tray, which has a context menu of a single item “Exit”.

All its functionality is accessed via a WCF remote interface (IDesktop). The methods provided by that interface are:

* LaunchProgram  
  Launch a predefined “player application” program by registered name, whose path and default arguments are configured in the “AvidConfig.xml” file. The program runs in the desktop, generally as full-screen and with access to the Windows sound output. Any other running program is closed or minimized as appropriate, so that only one player will run at a time.
* LaunchNewProgram  
  Launch a predefined “player application” program as above, without closing any other running program. This allows (for example) music to play when viewing photos.
* ExitProgram  
  Exit the named launched program if it is running.
* ExitAllPrograms  
  Exit any and all launched programs which are currently running.
* ForegroundProgram  
  Cause the named running program to come to the front of the desktop.
* SendKeys  
  Send a sequence of keystrokes to the foreground running application as though they had been typed on the keyboard.
* SendSpecialkey  
  Send one of a number of predefined “special” keys (e.g. a Windows key combination such as “Minimize” or “Context Menu”) as though it had been pressed on the keyboard.
* MouseMoveRelative  
  Move the mouse cursor by an amount relative to its current position as though a mouse had been moved.
* MouseClick  
  Send a mouse click to the foreground running application as though a mouse button had been pressed.
* SendIR  
  Send a pre-recorded Infrared code through the attached IR transmitter device. This is only used to turn the Sky set-top box on and off and to send lesser-user controls to the Sky box.
* FetchCoreTempInfoXml  
  Return an XML string containing the temperature and load statistics for each CPU core and the GPU. This information is currently not displayed through any UI, as over-heating is not an issue with the latest Avid media PC hardware.
* EnsureRemotePotatoRunning  
  Ensure that the RemotePotato service (described later) is still running and if has unexpectedly died, restart it.

## Sky Locator

Sky set-top boxes have a usable but undocumented and unsupported web service interfaces for all the common actions. This interface is accessed via a set of LAN URLs that can only be obtained by the discovery protocol SSDP. The process environment in an IIS worker process (where the Sky controller component runs) does not have the necessary privileges to use SSDP, which involves LAN multi-cast transmission. Consequently another process is needed for this. Avid.Desktop already has these privileges, as it runs as the system administrator user. So for convenience, when the Avid.Desktop tray application starts, it run the “SkyLocator” class methods to discover the Sky web service URLs for a Sky box on the local LAN, which it stores in the registry. Later when the Sky controller needs to use the web services, these URLs are already discovered and are available for immediate use.

# Avid.Spotify

Avid.Spotify is a full-featured Spotify player and play queue manager with no user interface, which can only be accessed via its web service interface. Although designed for use within Avid, it is self-contained and may be useful in other applications. Note it has no capability for browsing Spotify artists, albums and tracks, which can be done by the Spotify Web API. But the Spotify Web API cannot be used to play music. However, this player does support the authentication user interaction necessary to use that Spotify Web API in another module.

Avid.Spotify runs with administrator rights in the logged-in user’s desktop session. It has no user interface other than a single icon in the system tray, which has a context menu of two items: “Exit” and “Login”. Both of these exit the tray application. On first execution, and also when starting following a “Login”, the tray application prompts for Spotify credentials (user name and password), which must be for a “Spotify Premium” account. Lesser accounts are not supported by the Spotify programming interfaces.

All its functionality is accessed via an MVC WebAPI remote interface. The methods provided by that interface are arranged into a number of “controllers”. Each method has a distinct HTTP URL with any required arguments passed as HTTP parameters.

The controllers are:

* PlayQueue  
  with methods to queue a track or complete album for playing, to identify the currently playing track, to fetch the complete current queue, to remove a track from the queue and to skip to a particular track position within the queue.
* Player  
  with methods to start, pause, continue and stop the player, to get the current position within the playing track, and to seek to a particular position within the playing track.

The implementation for the Spotify access is via the libspotify native library. This is “wrapped” into the .Net execution environment by the SpotiFire open source library.

Across the MVC WebAPI remote interface, tracks, albums and artists are identified by the appropriate Spotify URI string.

The actual playing of each track is handled by the NAudio open source library. When Spotify is playing a track, it receives batches of playable audio samples from the Internet via asynchronous events. These samples are passed directly to the NAudio buffer (of five seconds capacity) which is output in the background to the Windows DirectSound output device. The buffering of these samples also maintains position counters within the track. Reaching the end of the samples for the current track is also notified by an event, which causes the next queued track to start playing, resetting the position counters.

## Spotify Web Authentication

The Spotify Web API is used by the Spotify class (described separately below). That class runs in an environment when no user interaction is easily possible. But the Spotify Web API uses NAUTH authentication that requires, through a browser page, for the user to log in to Spotify and approve the Avid4 use of the API.

In order to allow that user interaction, the Avid.Spotify tray application performs the authentication via NAUTH and obtains a “Refresh Token” string that can be used (without further interaction) to re-authenticate the Avid4 use. This “Refresh Token” is stored as a string value in the registry. Later when the Spotify class needs to use the web services, this persistent “Refresh Token” string is used to authenticate the current session.

## Spotify Web Authenticator

The NAUTH authentication described above requires public web URLs which can be used by the Spotify web APIs to perform the handshake. This is a simple MVC4 web application, following the NAUTH2 protocol, and in addition has actions to authenticate based on the “Refresh Token” and for the Spotify Web Authentication above to fetch that “Refresh Token” to be persistently sored in the registry.

The code embeds a hosted URL and Client ID string which are registered with Spotify for use by an application named “Avid4 Spotify Browser”. The authenticator code also embeds the “secret” for that application. When adapting the software for other use, the URLs, Client ID and secret should come from another registered application and the authenticator should be separately hosted on the public internet.

# Avid4G.Net

The Avid4G.Net web application follows the Model-View-Controller (MVC4) structure. There are no explicitly implemented models. Instead the model presented is handled by a set of middleware classes each of which interfaces with a particular player application or hardware component.

Each set of related views for a running application is implemented as a single MVC4 controller. Each controller has a set of web views. These sets of views all have two variants: wide and narrow. The wide views are intended for larger screen devices, such as tablets. The narrow views are intended for smaller screen devices such as smart phones. Buttons which navigate between views are constrained to link within the same variant: wide to wide and narrow to narrow. This preserves the overall user experience appropriate for each device.

The views are arranged as panes, each of which is independently scrollable. The narrow views normally have only a single pane. The wide views can have multiple views, often three. In a wide view, library browser type panes are generally positioned at the left of the view, and player transport buttons are generally positioned at the bottom right. This layout tends to suit a right-handed user best.

Each controller has an associated JavaScript file included in each of its views. That file implements the response of the UI elements (lists, buttons etc) to user interactions (tap, hold, drag, swipe etc). This response could be one or more of:

* Acting purely within the UI with no effect on Avid (e.g. scrolling).
* Invoking some Avid action through an AJAX request to a Controller URL (e.g. pausing a player).
* Replacing the contents of a single pane (partial view) within the view displayed in the browser (e.g. browsing the albums of a selected artist).
* Totally replacing the complete view within the browser, probably with a complete change of the JavaScript (e.g. launching a totally new player).

## MVC4 Controllers and Views

### Home and Top Bar

In the narrow variant, the Home view allows selection of a player application. It contains buttons for “TV”, “Radio”, “Sky”, “Video”, “Music”, “Spotify”, “Photos”, “Web and “Guide”. Each of these launches an appropriate player and switches to the default view for the appropriate controller.

In the wide variant, the Home view is blank.

All views, including the home views, contain a “top bar” partial view of a fixed set of controls that appear at the top of all views. The colour of this top bar is styled to indicate the current player application.

These controls include:

* A “Tool” button for special actions and non-default behaviour.
* A Volume display
* Volume buttons (increase, decrease, and mute toggle)
* A Home/Back/Off button, which either displays the Home view or (when on the Home view) turns off all player applications.

In addition, in the wide variant, the top bar contains the player application buttons, which are then accessible from all views without needing to access Home first.

The top bar (and therefore every view) has a JavaScript handler function (SwitchPanelAfterWake) that determines if it is being viewed for the first time in the last minute. This case will occur when (for example) a new controlling device is awoken from its sleeping state. When this case occurs, the view is automatically switched to a suitable default view for the currently running player application, as the view displayed when the device last updated (i.e. before sleeping) may no longer be appropriate.

The “Tool” button available on all screens displays an overlaid “menu” of less-needed actions:

* Screen off
* Screen on
* Visual on
* Log Fire
* Sound on TV
* Sound in rooms
* Mouse etc
* Load new media
* Restart Avid

Turning the screen on and off is normally handled automatically according to the selected player. But it can be controlled manually as well.

Visuals are the display of “music visualization” for Music and Spotify using the G-Force visualization software. The screen is not normally turned on by default for these players, but can be done so manually for a responsive pretty display to accompany the music.

A video of a Log Fire which can be displayed (silently) on its own or in addition to Music or Spotify players.

The routing of the output sound is normally handled automatically to suit the player application. If the TV screen is used, the sound uses the TV speakers. When no screen is used (e.g. Radio, Music and Spotify), the higher quality music speakers around multiple rooms are used instead. This default choice can also be overridden manually.

Occasionally it is necessary to interact with an on-screen display. It is the nature of Windows and some of the player applications that they occasionally need to interact with users, expecting a keyboard and mouse. To avoid the need to use a real (wirelessly connected) keyboard and mouse, the Tools menu can display a touch controlled “mouse pad” with a few control keys. This is almost always sufficient.

Avid’s media database must be manually reloaded when new music, photos, videos are added externally.

The Avid software can always be manually restarted in the event of unexpected failure. This is rarely needed.

### Action

The ActionController implements web methods for actions which are not interacting with any particular player application. These methods include those for launching and existing player applications, controlling volume, and the actions of the common top bar Tools menu. The ActionController URLS are:

* /Action/GetRunning [AJAX query]
* /Action/VolumeUp [Action]
* /Action/VolumeDown [Action]
* /Action/VolumeMute [Action]
* /Action/VolumeGet [AJAX query]
* /Action/Launch [Action]
* /Action/StartSky [Action]
* /Action/AllOff [Action]
* /Action/MouseMove [Action]
* /Action/MouseClick [Action]
* /Action/SendKeys [Action]
* /Action/SendIR [Action]
* /Action/ScreenOff [Action]
* /Action/ScreenOn [Action]
* /Action/VisualOn [Action]
* /Action/LogFire [Action]
* /Action/SoundTV [Action]
* /Action/SoundRooms [Action]
* /Action/RebuildMediaDb [Action]
* /Action/RecycleApp [Action]

### TV

The TvController implements views and action methods for live terrestrial TV and Radio played through the Windows Media Center player application. These URLS are generally used as actions by UI functions in tv.js. The TvController URLS are:

* /Tv/Watch [Narrow View]
* /Tv/ControlPane [Updatable Pane]
* /Tv/Channels [Narrow View]
* /Tv/ChannelsPane [Updatable Pane]
* /Tv/Radio [Narrow View]
* /Tv/RadioPane [Updatable Pane]
* /Tv/NowAndNext [AJAX query]
* /Tv/ChangeChannel [Action]
* /Tv/Action [Action]
* /Tv/Buttons [Narrow View]
* /Tv/All [Wide View]

### Sky

The SkyController implements views and action methods for live and recorded TV and Radio played on an external Sky set top box. These URLS are generally used as actions by UI functions in sky.js. The SkyController URLS are:

* /Sky/Watch [Narrow View]
* /Sky/ControlPane [Updatable Pane]
* /Sky/Live [Narrow View]
* /Sky/ChannelsPane [Updatable Pane]
* /Sky/Radio [Narrow View]
* /Sky/RadioPane [Updatable Pane]
* /Sky/NowAndNext [AJAX query]
* /Sky/Recordings [Narrow View]
* /Sky/RecordingsPane [Updatable Pane]
* /Sky/Recording [Narrow View]
* /Sky/RecordingDescription [AJAX query]
* /Sky/Buttons [Narrow View]
* /Sky/All [Wide View]
* /Sky/ChangeChannel [Action]
* /Sky/PlayRecording [Action]
* /Sky/DeleteRecording [Action]
* /Sky/Play?speed=NNN [Action]
* /Sky/Pause [Action]
* /Sky/Stop [Action]

### Video

The VideoController implements views and action methods for recorded TV ripped DVDs played through the Zoom Player player application. These URLS are generally used as actions by UI functions in video.js. The VideoController URLS are:

* /Video/Watch [Narrow View]
* /Video/WatchPane [Updatable Pane]
* /Video/All [Wide View]
* /Video/Recordings [Narrow View]
* /Video/Recording [Narrow View]
* /Video/DVDs [Narrow View]
* /Video/RecordingsPane [Updatable Pane]
* /Video/DVDsPane [Updatable Pane]
* /Video/PlayRecording [Action]
* /Video/DeleteRecording [Action]
* /Video/PlayDvdDisk [Action]
* /Video/PlayDvdDirectory [Action]
* /Video/PlayBluRayFile [Action]
* /Video/GetPlayingInfo [AJAX query]
* /Video/SendZoom [Action]

### Music

The MusicController implements views and action methods for stored music and music streamed from BBC iPlayer Radio played through the J River Media Center player application. These URLS are generally used as actions by UI functions in music.js. The MusicController URLS are:

* /Music/All [Wide View]
* /Music/Playing [Narrow View]
* /Music/Queue [Narrow View]
* /Music/QueuePane [Updatable Pane]
* /Music/Browser [Narrow View]
* /Music/BrowserPane [Updatable Pane]
* /Music/GetPlayingInfo [AJAX query]
* /Music/SendMCWS [Action]
* /Music/RemoveQueuedTrack [Action]
* /Music/PlayListenAgain [Action]
* /Music/GetListenAgainIcon [AJAX query]

### Photos

The PhotosController implements views and action methods for stored photographs viewed through J River Media Center player application. These URLS are generally used as actions by UI functions in photos.js. The PhotosController URLS are:

* /Photos/Display [Narrow View]
* /Photos/Browse [Narrow View]
* /Photos/Images [Narrow View]
* /Photos/ImagesPane [Updatable Pane]
* /Photos/All [Wide View]

### Spotify

The SpotifyController implements views and action methods for streamed music played through the Avid.Spotify player application. These URLS are generally used as actions by UI functions in spotify.js. The SpotifyController URLS are:

* /Spotify/All [Wide View]
* /Spotify/Playing [Narrow View]
* /Spotify/Queue [Narrow View]
* /Spotify/QueuePane [Updatable Pane]
* /Spotify/Browser [Narrow View]
* /Spotify/BrowserPane [Updatable Pane]
* /Spotify/GetPlayingInfo [AJAX query]
* /Spotify/PlayAlbum [Action]
* /Spotify/PlayTrack [Action]
* /Spotify/SkipToQueuedTrack [Action]
* /Spotify/RemoveQueuedTrack [Action]
* /Spotify/PlayPause [Action]
* /Spotify/Skip [Action]
* /Spotify/Back [Action]
* /Spotify/Plus10 [Action]
* /Spotify/Minus10 [Action]
* /Spotify/SetPosition [Action]
* /Spotify/GetAlbumImage [AJAX query]
* /Spotify/AddTrackToPlayList [Action]
* /Spotify/AddAlbumToPlayList [Action]
* /Spotify/RemoveTrackFromPlayList [Action]
* /Spotify/RemoveAlbumFromPlayList [Action]

### Web

The WebController implements views and action methods for streamed BBC iPlayer TV played through the Internet Explorer browser acting as a player application. These URLS are generally used as actions by UI functions in web.js. The WebController URLS are:

* /Web/Mouse [Narrow View]
* /Web/All [Wide View]
* /Web/Browser [Narrow View]
* /Web/BrowserPane [Updatable Pane]
* /Web/PlayBBC [Action]

### Guide

The GuideController implements views and action methods for viewing the Electronic Programme Guide (EPG) and setting recording for terrestrial TV and Radio programmed though the RemoteMotato service which presenta an API onto data within the Windows Media Center. These URLS are generally used as actions by UI functions in guide.js. The GuideController URLS are:

* /Guide/Browser [Narrow View]
* /Guide/BrowserWide [Wide View]
* /Guide/BrowserPane [Updatable Pane]
* /Guide/Description [AJAX query]
* /Guide/Record [Action]
* /Guide/Cancel [Action]

## Middleware Classes

The “Model” aspect of the MVC pattern is provided by a set of “middleware classes”, each of which is designed to offer a useful API into a single player application, external service or hardware component. The controller and view classes implement the necessary UI views and actions though these middleware classes.

### JRMC

The JRMC class encapsulates all access to the J River Media Center player application which is used for cataloguing and playing all stored music, music streamed from BBC iPlayer Radio and for viewing photos.

The main interaction with the JRMC running as a service is its “MCWS” (media center web service) service running as an HTTP web service on port 52119 on the local host. This web service API is fully documented within the JRMC software.

While that API is very rich, its very generality does not efficiently handle the access patterns used within Avid.

The structuring of the Avid music catalogue splits music into “classical” and “non-classical”. Classical music is primarily indexed by composer, and then within composers by album. Non-classical music has two primary indexes – by “album artist” and by album name. When indexed by “album artist”, music is further keyed by album name. Both the “album artist” and “album name” primary indexes are also indexed by the initial letter. Finally the most expected style of music playing is that generally complete albums are played as a whole.

To support this indexing pattern, a custom structure is maintained by the JRMC class (of classes TrackData, AlbumData and AlbumCollection). This internal structure is constructed as an object hierarchy though a tree-walk of the J River data through the MCWS interface. But as the cost of constructing the object structure is high (tens of seconds), the complete structure is serialized out to a binary cache file. On startup, this file is loaded in preference to the tree-walk – a sub-second activity. When the J River data changes (e.g. when a new album is purchased), the LoadAndIndexAllAlbums method rebuilds the binary cache.

The structure and purposes of the JRMC class and its members should be fairly clear from the commenting. However, there a few aspects worth noting:

* In addition to the AlbumList (and related collections) there is a PhotoAlbumList used by the PhotoController and its views. This is constructed from MCWS in the same manner.
* There are some “hard-wired” queries that suit the album-oriented usage. These include GetLuckyDipAlbums and GetRecentAlbums, both of which return data on twenty albums.
* An album is determined as “classical” if its path contains the directory “\Classical\”. This then imposes no constraints on the genre or other meta-data tags.
* Some methods support streaming from a BBC iPlayer Radio URL instead of from stored music files. When streaming, the current title, album, artist and album image are faked up to data provided by the BBC class alongside the stream URL.
* JRMC has various screen modes and there are methods to set the appropriate mode for the running player application.

### Zoom

The Zoom class encapsulates all access to the Zoom Player application which is used for watching recorded terrestrial TV programmes, ripped and physical DVDs and ripped Blu-ray disks.

The class uses a combination of:

* Command-line parameters to invoke Zoom Player to play from a particular filename, directory or device in the appropriate mode.
* The web service on port 4768, which is used to send action (“fn…”) commands to a running Zoom Player instance. These are sent via the SendRequest method.
* The TCP port 4769 which is used to request and receive state data asynchronously from a running Zoom Player instance. This data is stored and made available through the GetInfo method and one or two other public properties.

The Zoom “fn…” codes are documented at:

* <http://www.inmatrix.com/zplayer/highlights/zpfunctions.shtml>

The structure and encoding of the data sent over the TCP interface are documented in:

* <http://forum.inmatrix.com/index.php?showtopic=7051>

We are only currently interested in a very small subset of the available codes.

One state change that is specifically tracked is the “Mode”, which can be one of “Media” (when watching a media file), “DVD” when a DVD is playing, and “Menu”, when the DVD is playing but is in menu-mode. This allows the Video view to dynamically switch between the transport controls (play, pause etc.) and the menu controls (arrows and OK) which can navigate the on-screen DVD menu.

### Sky

The SkyData class is designed to control and access a Sky Set Top Box via its (undocumented) web service interfaces.

The singleton static property Sky is used to access all members. Before accessing the Sky property, it is necessary to first call the Initialse() method.

The data made available by the SkyData class comes from a number of sources:

* Sky.com public web services lists all channels and provides an Electronic Programme Guide. The specific public services used in Avid are:
  + <http://tv.sky.com/channel/index>, which lists all currently broadcast channels with their internal and external identifiers.
  + <http://epgservices.sky.com/5.1.1/api/2.0/channel/json/CHANNEL/now/nn/4>, which returns Now-and-Next EPG data for the specified channel. Note that Avid does not display a full Sky EPG beyond Now-and-Next; there are many other good options for this.
* The Sky set top box implements a number of web services on its network connection. The URLs for these services are discovered by the SkyLocator class described earlier. The two specific local web services used are:
  + SkyBrowse, with SOAP methods to browse all recordings stored and to delete a particular recording.
  + SkyPlay, with a method to get information on what is currently playing, either a channel on live TV or a recording.

In addition the SkyPlay services has action methods to change the live channel, play a recording, Play, Pause, Fast Forward, Rewind and Stop the current live TV or recording.

Note that the Sky set top box web services are totally undocumented and unsupported. The current uses have been reverse engineered by packet sniffing similar capabilities in the Sky IOS apps, inferring the interpretation of the request and response data empirically, to reach a set of useful functions that work adequately.

There appears to be no public web service to list the Radio channels. So currently a set of these radio channels can be configured in the XML configuration accessed via Config class alongside the specification of “favourite” TV channels to be listed first. Also, the channels listed are filtered to contain only the set of free-to-air channels plus those for a configured set of subscription options.

### Spotify

The Spotify middleware class has quite distinct two capabilities each based on a different access mechanism. One is simply a .Net class wrapper around the Avid.Spotify WebAPI methods. These are organized in four groups:

* PlayQueue  
  with methods to queue a track or complete album for playing, to identify the currently playing track, to fetch the complete current queue, to remove a track from the queue and to skip to a particular track position within the queue.
* Player  
  with methods to start, pause, continue and stop the player, to get the current position within the playing track, and to seek to a particular position within the playing track.

Each method is identically implemented as an HttpClient request/response to the appropriate Avid.Spotify WebAPI URL.

But for browsing and management of playlists, methods use the Spotify Web API accessed through the SpotifyAPI-Net wrapper library from (<https://github.com/JohnnyCrazy/SpotifyAPI-NET>).

* Browse  
  with methods to search for artists, albums and tracks and to fetch albums for an artist, tracks for an album, similar artists, etc. This allows usable navigation of the Spotify catalogue.
* PlayList  
  with methods to list the Spotify account’s playlists and to enumerate the tracks and albums of these existing playlists. Further method allow playlists to be created, renamed and deleted, and for tracks and albums to be added to and removed from playlists, and to access and update “Saved” albums.

### RemotePotato

RemotePotato is a free, recently open sourced, Windows service available for download from <http://www.remotepotato.com>. Its main purpose is to present a browser view onto the Windows Media Center (WMC) EPG and scheduled recordings. This information is then available from any browser either as Silverlight or a more simple HTML view.

Crucially for Avid, the RemotePotato service also has a documented Web API, accessible via an HTTP socket, which implements a relatively full-featured remote interface onto the functionally of WMC. As well as its access to the EPG and scheduled recordings, the Web API provides direct control of a running WMC through keyboard emulation.

The Avid RemotePotato middleware class accesses this Web API, thus providing access to much of the functionality of WMC. This is used for:

* Control of a running WMC for viewing live TV and listening to live radio though terrestrial or satellite tuners via the TV MVC controller.
* Scheduling of recordings based on the EPG via the Guide MVC controller.
* Discovery and identification of existing recordings for viewing though another media player (in practice Zoom Player) via the Video MVC controller.

Through the RemotePotato service, the RemotePotato middleware class implements members to:

* List all TV and radio channels.
* Get Now-and-Next programme information for a channel or a full EPG for a channel for a specific day.
* Select a channel to view or listen to, noting the current channel.
* Control (play pause, rewind, fast forward, stop) the viewing of live TV.
* Get the set of existing recordings, with information (including stored file path) for each recording.
* Schedule and cancel recording of future TV and Radio programmes.

### BBC

The BBC class can obtain the historical schedule of BBC Radio or TV programmes and iPlayer URLS from which Radio can be streamed or TV can be watched.

The set of BBC TV and Radio channels of interest is configured in the AvidConfig.xml file, which for each station configures the display name and the internal BBC identifier. When a channel and date are selected though the UI, the GetSchedule method reads the historical schedule of programmes broadcast (on that date on that channel) from the BBC web service URL <http://www.bbc.co.uk/CHANNELID/programmes/schedules/DATE.xml>.

Other methods, given the “Programme ID” (Pid) for a programme selected from the schedule, can return URLs. One URL is of streamable WMA data for a radio programme. This URL can then be played by the J River Media Center. Note that the WMA stream is not of the highest quality, but the higher quality BBC streams cannot be played in external players. The other URL, for a TV program, is of a browser page on which the programme may be viewed in full screen. Actually watching the streamed programme requires interaction with the browser through mouse emulation.

### Running

The Running middleware class is responsible for launching and exiting player applications, for ensuring that two players are not running at the same time, and for each application ensuring that the screen is turned on or off as appropriate, and that the sound is routed to the correct set of speakers by default. It keeps track of the currently running player application.

### Desktop

The Desktop middleware class is simply a .Net class wrapper around the Avid.Desktop WCF methods. These methods are responsible for launching and controlling player applications, together with other miscellaneous simulated user input functionality.

### Receiver

The Receiver middleware class provides methods for querying and controlling a Yamaha AV receiver. The only model with which this has been used is the RX-V671, but I strongly suspect that it will work with other models in the same range. The significant aspect of the Yamaha receiver is that it can be controlled via an easy to understand HTTP protocol.

The receiver (at least that particular model) has two separate zones. One is digital and can route digital inputs (HDMI or SPDIF) to a 5.1 speaker array. The other zone is analog only and can route analog inputs to stereo speakers and auxiliary stereo amplifiers in other rooms.

It is a constraint of the Yamaha connectivity that the digital inputs cannot be used as inputs to the stereo speakers and auxiliary stereo amplifiers.

Consequently each connected input device (the media PC and the Sky set top box) are connected to the receiver by **both** digital and analog audio connectors, each carrying the same audio content. Consequently, with this redundant wiring, the input source and output speakers can be independently controlled by software over the HTTP protocol.

The Receiver middleware class has methods to:

* Turn on and off the two “output zones” (i.e. two sets of speakers) independently. Normally only one zone will operate at any one time.
* Control the volume and muting of the “current” zone.
* Specify the input source (paired analog and digital connectors from media PC or Sky).

### Screen

The display screen for Avid is a high quality Samsung Smart TV, connected as the HDMI output of the Receiver. However, most of the capability of this TV is wasted as it is simply used as a 1920x1080 output screen for any media PC player application (or Sky box) that needs a visual display. None of its TV tuner or “smart” networked capability is used.

The only control for the screen is to be able to turn it on or off.

Unfortunately, the Samsung cannot be controlled by its network interface. There is no network control service running on the TV firmware, and experiments have shown that the network port is totally powered off when the TV goes into standby!

The power can be controlled via the HDMI-CEC protocol, using only two CEC commands – one to turn on the screen, and one to turn it off. The HDML-CEC codes can be generated by an external RainShadow USB HDMI-CEC bridge (<http://rainshadowtech.com/HdmiCecUsb.html>), with the ValkyrieTech (<http://valkyrietech.com/>) HTTP service as an easily accessible API. However, that mechanism has the complication that the state of the TV screen is also sensed by the receiver via the HDMI cable. The receiver interprets the screen turning on as desire to set the audio input to that of the TV HDMI itself. No totally reliable remedy for this mis-interpretation has been found. Consequently the RainShadow device is no longer used.

Control by infra-red is possible, but problematical as the only standard IR code is a power toggle. However, discrete codes for Samsung TVs are unofficially documented, and the discrete “Power On” and “Power Off” codes have been found to be reliable. So infra-red is the preferred mechanism to turn the screen on or off.

There is one additional complication arising when the screen is turned on

* The “full screen” views of some of the media PC applications interrogate the current screen when they start. Until the screen is turned on, the display characteristics seen by the player application will be incorrect, leading to incorrect output. Consequently it is sometimes necessary to wait for the screen to be seen to be on (by the media PC), and there is a method for this.

### IRCodes

Avid.Desktop has a capability of transmitting recorded infra-red code sequences to control external devices. In Avid4, the only devices that can be controlled by IR that do not have a more suitable network control capability are the Sky set-top-box and the Samsung TV screen – and then only for turning it on and off and for less common Sky controls.

The IRCodes middleware class contains the definitions of the set of recorded infra-red code sequences, each associated with a symbolic name. These code strings can be looked up (by symbolic name) in the Codes Dictionary in the class, to be sent to Avid.Desktop for transmission.

### Config

The Config middleware class reads a single XML file from the path “C:\Avid.Net\AvidConfig.xml”. The values in that XML files can be used to configure paths, IP addresses, favourites etc. The properties of the Config class are:

* IpAddress - The Media PC's fixed IP address
* ReceiverAddress - The Receiver's IP address
* VideoPath - The path to the directory in which recorded TV programmes are stored
* DvdPath - The path to the directory in which ripped DVDs are stored
* BluRayPath - The path to the directory in which ripped BluRays are stored
* TvFavourites – A collection of favourite terrestrial TV channels to be displayed first in any lists
* SkyFavourites -A collection of favourite Sky channels to be displayed first in any lists
* SkyRadio - A collection of Sky Radio channels
* SkyPackages - A collection of subscribed Sky Package codes
* SkyCapacityGB - The capacity of the Sky box for recordings (in GB)
* UseSkyLiveTV - When true, indicates that Sky should be used for preference over the terrestrial tuners for live TV
* BBCTVChannels - The collection of BBC TV channels to be made available for iPlayer
* BBCRadioStations - The collection of BBC radio stations to be made available for iPlayer

Notes:

* If ReceiverAddress is omitted from the AvidConfig.xml, the software will operate without attempting to control the receiver. This allows testing on a development machine without affecting the live Avid that may be operating on the same LAN.

The AvidConfig.xml file also contains <Program> elements in which the paths and optional arguments for the named player applications are specified. The XML is also read by Avid.Desktop for these values. The named player applications with <Program> elements are:

* TV – Windows Media Center
* Web – Browser for iPlayer
* Video – Zoom Player
* Photo – J River Media Center launcher application for photographs
* GForce – Stand-alone visualization programme for use when Spotify is playing
* LogFire – A separate invocation of Zoom Player, which plays a fixed “Log Fire” video

### Render

The Render middleware class is used as a utility class in MVC views to construct the HTML <script> tag with the URL of a JavaScript file, such that the file is cached only until the source file changes. This copes with the development cycle when the JavaScript file may change at any time, while still benefiting from the caching of an unchanged file in normal operation. It works by appending an unused argument to the generated script URL whose value is the timestamp of the real source file for the JavaScript, so that changes in the source file result in different (separately cached) URLs.

## User Interface

The user interface for controlling Avid is purely via HTML in a web browser on a touch device (smartphone or tablet). The HTML is generated as MVC views using the “Razor” view engine notation, which combines C# code and HTML in an easily understood manner. The C# code accesses data to display from the appropriate middleware classes.

The views come in two variants: wide and narrow. The wide views are intended for larger screen devices, such as tablets. The narrow views are intended for smaller screen devices such as smartphones. Buttons which navigate between views are constrained to link within the same variant: wide to wide and narrow to narrow. This preserves the overall user experience appropriate for each device.

The variant views are then assembled from included “pane” partial views. A narrow view will normally have a set of buttons to switch panes and then display only a single selected pane. A wide view will normally display all the panes appropriate for the current player application – usually three. By convention, in a wide view:

* The complete left-hand-side is a “browsing” or “selecting” list of things to play.
* The lower right-hand-side contains the play controls.
* And the upper right-hand-side contains either a list of things playing or some additional controls.

The top of every view is the common “top bar” discussed earlier – identical (apart from colour) in every view for every player application.

The Razor view engine generates “pure” HTML with no embedded styling or JavaScript. Objects in the HTML all have class and id names as necessary to label the content for CSS styling or handling touch actions. A single CSS file handles all the styling of all views in a consistent manner. Each player application has a single JavaScript script file included to handle the user interaction behaviour for that player. In addition, a common JavaScript file (Functions.js) contains some functions and behaviours common to multiple players.

The JavaScript which implements the user interactions does **not** use a touch-centric platform, such as JQuery.Touch or JQuery.Mobile. Experience has shown that these are nowhere near responsive enough for the requirements of Avid.

The JavaScript, does however make extensive use of JQuery, which significantly enhances the maintainability of the script code. All the touch handling (touch, tap, doubletap, hold, drag, swipe, pinch) is handled by Hammer.js (<http://eightmedia.github.io/hammer.js/>). Each pane has a Hammer object with all necessary actions bound to appropriate buttons or list items. This has proven very effective in constructing a consistent set of responsive behaviours.

In addition, Functions.js implements two common touch behaviours:

* EnableDragScroll, when added to a Hammer object for a pane of list items (e.g. album titles), makes that list touch-scrollable within a fixed area of the browser.
* EnableMouseBehaviour, when added to a Hammer object for a blank rectangular area, makes that area act like a mouse touch-pad, sending MouseMove and MouseClick actions as the area is touched, dragged and tapped.

Additional third-party JavaScript used are:

* noUiSlider (<http://refreshless.com/nouislider/>) – a lightweight and minimal on-screen slider control.
* Jquery.easing (<http://gsgd.co.uk/sandbox/jquery/easing/>) – which adds attractive deceleration to the touch drag and swipe behaviours.

The buttons are all constructed as PNG images, maintained in Photoshop. At some point I will re-draw then to a higher resolution, but they are fine for now.

# External Hardware and Software

## Hardware Devices

The Avid4 media PC has only one non-standard device. It needs its own specific device control software (drivers etc).

### USB UIRT

To control the Sky box, for those aspects not supported by the Sky web services, the only mechanism is Infra-red. This is particularly necessary for turning the Sky box on and off. This is achieved using a USB-UIRT device, which is an infra-red transmitter, connected via USB, obtained from <http://www.usbuirt.com/>. This is used via the UsbUirtManagedWrapper.UsbUirt class in the UsbUirtManagedWrapper DLL included in ReferencedDLLs.

The Infra-red codes for the Sky box (and other devices) were originally recorded using Girder (<http://www.promixis.com/girder.php>) which was a component of the earlier implementations Avid1 and Avid2. IR codes for many devices are available on the web, but with the correct software, the USB-UIRT device can easily record codes.

## Commercial Software

A number of commercial player applications are used for the various media types. Generally each application is the “best of kind” for the content that Avid uses it for. An additional factor in the choice of player is the ability to control it via a network. It is highly likely that many player applications could replace the functionality of the others. But the mix of player applications used has proven easy to control though usable network protocols.

### Windows Media Center

Microsoft Windows Media Center (WMC) is used for live terrestrial TV and Radio, for watching, listening and recording. This software comes bundled with a number of variants of Windows 7, and is an inexpensive add-on to Windows 8 Pro.

This player application is used in Avid4 for:

* Watching Live TV (with the ability to pause live TV)
* Listening to Live Radio
* Recording TV and Radio from an EPG schedule.

Access to the programming interface of WMC is provided by RemotePotato (described below),

A configuration option allows live TV and Radio to come from Sky in place of WMC. This will generally be higher quality than terrestrial, but requires one of the two Sky tuners which could otherwise be used for recording.

### J River Media Center

J River Media Center (<http://www.jriver.com/>) is used to play stored music, to view stored photographs and to listen to streamed BBC iPlayer Radio.

### Zoom Player

Zoom Player (<http://inmatrix.com/>) is used to play TV recorded (by WMC), ripped DVDs, Ripped Blu-ray disks and physical DVD disks.

### Spotify

The Spotify player application is **not** used to play music streamed from Spotify. But the Avid.Spotify player service **does require** a current Spotify Premium account. It also requires a Spotify “developer key”, freely available with a premium account.

### G-Force

My preferred music visualization software is G-Force from SoundSpectum (<http://www.soundspectrum.com/>). This integrates into J River Media Center, and will display in that player’s “Display View”. The Platinum version of G-Force (or other SoundSpectum visualization products) comes with a stand-alone player that can be executed to generate visualizations to accompany streamed Spotify music.

## Free and Open Source Software

### RemotePotato

RemotePotato is a free, recently open sourced, Windows service available for download from <http://www.remotepotato.com>. Its main purpose is to present a browser view onto the Windows Media Center (WMC) EPG and scheduled recordings. This information is then available from any browser either as Silverlight or a more simple HTML view.

Crucially for Avid, the RemotePotato service also has a documented Web API, accessible via an HTTP socket, which implements a relatively full-featured remote interface onto the functionally of WMC. As well as its access to the EPG and scheduled recordings, the Web API provides direct control of a running WMC through keyboard emulation.

### JQuery

The JavaScript code for the UI makes extensive use of the JQuery library (<http://jquery.com/>). It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

### Hammer.js

All the touch handling (touch, tap, doubletap, hold, drag, swipe, pinch) is handled by Hammer.js (<http://eightmedia.github.io/hammer.js/>). This lightweight and efficient JavaScript adds a rich touch capability of a browser-based UI, without the complexity and significant performance costs of adopting a library such as JQuery Mobile.

### noUiSlider

The sliders used for progress bars etc are implemented using noUiSlider (<http://refreshless.com/nouislider/>) – a lightweight and minimal on-screen slider control. This takes the form of a JavaScript library for the slider behaviour and a CSS style sheet.

### SpotiFire

The Spotify music streaming service can be accessed (by Premium subscribers) though a library of API methods. SpotiFire (<https://github.com/Alxandr/SpotiFire>) is an open source collaborative project which wraps that library into C# .Net classes in a manner suitable for use in Avid code. The Avid.Spotify player uses SpotiFire extensively.

### SpotifyAPI-Net

The Spotify Web API can most easily be accessed from .Net code by a wrapper library. The use used in Avid4 within the Spotify middleware class is SpotifyAPI-NET found at <https://github.com/JohnnyCrazy/SpotifyAPI-NET>, developed under the LGPL open source licence.

### NAudio

NAudio (<http://naudio.codeplex.com/>) is an open source .NET audio and MIDI library, containing dozens of useful audio related classes intended to speed development of audio related utilities in .NET. Its audio output functions are used in the Avid.Spotify player to play Spotify music.

### NLog

NLog (<http://nlog-project.org/>) is a free open source logging platform for .NET and other platforms. It is used in Avid.Desktop and Avid.Spotify, whose App.config files configure logging to files in the directory “C:\Avid.Net\Logs". And it is used in Avid4G.Net, whose Web.config file configures logging to files in the sub-directory “Logs” within the IIS web application directory. In both cases, a new log file is started each day, named by the date.

### InputSimulator

The Windows Input Simulator (<http://inputsimulator.codeplex.com/>) provides a simple .NET (C#) interface to simulate Keyboard or Mouse input using the Win32 SendInput method. This is only used for more complex keypress simulations in Avid.Desktop.DesktopService.SendSpecialkey that cannot be handled by System.Windows.Forms.SendKeys.SendWait. This service is used via the WindowsInput.InputSimulator class in the InputSimulator DLL included in ReferencedDLLs.

### Core Temp

Core Temp (<http://www.alcpu.com/CoreTemp/>) is a compact, no fuss, small footprint, yet powerful program to monitor processor temperature and other vital information. Although the current UI views do not display this information, it is monitored and available for display. This service is used via the GetCoreTempInfoNET.CoreTempInfo class in the GetCoreTempInfoNET DLL included in ReferencedDLLs.

### GPUZ

GPU-Z (<http://www.techpowerup.com/gpuz/>) is a lightweight system utility designed to provide vital information about your video card and graphics processor. The information from this utility supplements that from Core Temp, and would normally be displayed in a similar manner. The current UI views do not display this information.

# Components with Other Potential Uses

Although Avid has been designed as a coherent whole (or has evolved over four major iterations to **appear** to have a common overall design), there are a number of components which may play a useful part in other software systems. Where possible, these component have been designed with this in mind, to allow the necessary code to be easily isolated and copied elsewhere. Note that the MIT open source license places no restrictions on using these components in any other manner.

## Spotify

The Avid.Spotify player application is a full-featured Spotify player and play queue manager with no user interface, which can only be accessed via its web service interface. This can certainly be used in isolation.

In addition there is a single C# source file (Spotify.cs) containing a Spotify client class for this web service interface, together with wrappers for the Spotify Web API used for browsing the Spotify catalogue and for playlist management. This client class makes available the functionality to .Net programs. But the structure of this client class is very simple and each method is a wrapper to an HTTP request. Consequently, it would be very easy to re-implement this wrapper for any other platform – including directly in JavaScript within a browser.

## Sky

Sky set top boxes have an unsupported and undocumented network web service interface. These services provide methods:

* To browse all recordings stored and to delete a particular recording.
* To get information on what is currently playing, either a channel on live TV or a recording.
* To change the live channel, play a recording, play, pause, fast forward, rewind and stop the current live TV or recording.

In addition, sky.com web services on the public internet provide mechanisms to list all available TV channels and provide an Electronic Programme Guide.

The middleware class SkyData is designed to encapsulate these web services in a simple wrapper, and has been designed to be easily isolated.

Note that the Sky set top box web services are totally undocumented and unsupported. The current uses have been reverse engineered by packet sniffing similar capabilities in the Sky IOS apps, inferring the interpretation of the request and response data empirically, to reach a set of useful functions that work adequately.

The LAN URLs for the set top box web services can only be obtained by the discovery protocol SSDP. The separate SkyLocator class (in the source file SkyLocator.cs within Avid.Desktop) has a single public method GetSkyServices to discover the Sky web service URLs for a Sky box on the local LAN, which it stores in the registry. Later when the SkyData class needs to use the web services, these URLs are already discovered and are available for immediate use.

## Hammer.js behaviours

The common JavaScript functions library in the source file Functions.js has a couple of methods that can be used to add functionality to other touch-based UIs based on Hammer.js.

* EnableDragScroll   
  For a Hammer.js object attached to a pane (a <div>), interpret vertical dragging and swiping as scrolling the contents of that pane with an attractive “easing” response.
* EnableMouseBehaviour  
  For a Hammer.js object attached to a pane (a <div>), interpret dragging and tapping within the pane as though it were on a touch pad or laptop mouse pad, sending mouse move and mouse click events.