# Architecture

## Application

The Artisan **Application** is the main Android activity that presents a user interface and ties everything together.

It consists of a number of Views, each of which is basically a separate activity, or set of functionality, with its own full page UI.

All communication between Views takes place through events sent to, and dispatched by, the Application.

Each View is generally associated with a **Device** which it presents, and allows the user to interact with.

In addition to the UI and it’s views, the Application is responsible for starting one or more **Servers** which are generally faceless, although they may interact with devices and their services, and thus impact the UI.

## Main Views (Activities)

These are the main “full-page” views in Artisan, without listing their interdependencies or dependencies on specific Objects.

### NOW PLAYING (aPlaying)

The Now Playing View presents a UI with transport and other controls to allow the user to control a **MediaRenderer** device**.**

MediaRenderer devices present at least one **Service**, an **AVTransport**, which can play a **Track**, stop, pause, etc.

Some MediaRenderer devices may present an additional **RenderingControl** service, which can control the volume, balance, fade etc.

Some MediaRenderer devices may present an additional **Playlist** service, which allows the control point to navigate thru a list of songs.

Some MediaRenderer devices may present an additional **PlaylistSource** service, which allows the control point to select a Playlist from a list of Playlists, save a named Playlist etc.

When a device does not support a Playlist service, Artisan will make use of the **LocalPlaylist** service to *drive* the more basic DLNA MediaRenderer.

When a device does not support a PlaylistSource service, Artisan will provide the **LocalPlaylistSource** service to provide the additional functionality of switching to named playlists, saving playlists, etc.

### LIBRARY (aLibrary)

The Library View presents a UI that allows a user to interact with a **MediaServer** device, which presents a **ContentServer** service, which allows the control point to navigate a tree of Folders and Tracks and play them, enque them to a playlist, etc.

### PLAYLIST (aPlaylist)

The Playlist view is generally associated with the Playlist and PlaylistSource services of the current MediaRenderer, with the note above that Artisan will provide them if needed.

The Playlist View allows a user to interact with the Playlist, for example to let the user select the next song for the renderer to play, clear it, save it, etc.

### EXPLORER (aExplorer)

An Explorer is a specialized hierarchical view which exposes inner details of the **Database** associated with a MediaServer/ContentServer.

The Explorer View is only enabled for the **LocalMediaServer** and **RemoteMediaServer** devices, and their associated ContentServers.

Note that a Database is distinctly different than a Library!!

### PREFERENCES (aPrefs)

allows the user to configure the application, choose renderers, libraries, set defaults for these, etc.

## Base Objects

These objects are shared throughout the system

### Folder

### Track

## Device and Service Base Classes

The main devices and device.services have abstract base classes that allow the rest of the system to deal with the devices and service polymorphically.

### MediaRenderer

#### AVTransport

#### RenderingControl

#### Playlist

#### PlaylistSource

### MediaServer

#### ContentServer

#### Database

## Concrete Devices and Services Classes

This is the hierarchy of *Client* devices and services which can be associated with the Artisan Views.

### device LocalMediaRenderer

#### LocalAVTransport

#### LocalRenderingControl

#### LocalPlaylist

#### LocalPlaylistSource

### device LocalMediaServer

#### LocalContentServer

#### LocalDatabase

### device RemoteMediaRenderer

#### RemoteAVTransport

#### RemoteRenderingControl

#### RemotePlaylist

#### RemotePlaylistSource

### device RemoteMediaServer

#### RemoteContentServer

#### RemoteDatabase

### device DlnaMediaRenderer

#### DlnaAVTransport

#### DlnaRenderingControl

#### \* LocalPlaylist

#### \* LocalPlaylistSource

### device DlnaMediaServer

#### DlnaContentServer

### device OpenMediaRenderer

#### \* DlnaAVTransport

#### \* DlnaRenderingControl

#### OpenPlayList

#### \* LocalPlaylistSource

## Other things in the prh.device folder

### SSDPDevice

Possibly unused

A class representing an SSDP device

### SSDPSearch

A class that does an SSDP Search for interesting devices

Events Artisan or creates Device/Service instances.

## Servers

Artisan starts one or more servers.

Most are self-contained single java classes.

The HTTPServer is a hierarchy of HTTP request handlers that correspond to the services presented by THIS device.

Note that these correspond to control of the LOCAL DEVICE only.

We do not support “pass-thru” control of other devices.

We do not currently support the ConnectionManager service.

My av\_transport, rendering\_control, and content\_server do not currently handle SSDP Event subscription or dispatching.

### HTTPServer

The HTTP Server presents the following paths

Each root path can be turn on or off by a preference (i.e. Advertise DLNA Server=true).

#### av\_transport

##### control

##### event

#### rendering\_control

##### control

##### event

#### content\_server

##### control

##### event

#### open\_home

##### product

###### control

###### event

##### playlist

###### control

###### event

##### info

###### control

###### event

##### volume

###### control

###### event

##### time

###### control

###### event

#### remote

##### av\_transport

##### rendering\_control

##### content\_server

##### open\_home?

##### database

### SSDPServer

The SSDP Server presents a list of possible (local) devices and services that this Artisan can present.

There is generally an HTTP receiver path, above, for each service it presents

#### SSDP uuid

#### SSDP rootdevice

#### DLNA MediaServer

##### ContentDirectory

#### DLNA MediaRenderer

##### AVTransport

##### RenderingControl

#### OpenHome Source

##### Product

##### Playlist

##### Info

##### Volume

##### Time

### Other things in the prh.server folder

#### LocalVolumeFixer

Lives in the server folder because it’s long lived, like the HTTPServer.

#### StartAtBoot

Just cuz it doesn’t look good in the main artisan folder

## Utility Classes

### Utils

### dlnaUtils

# Implementation

## Initial Implementation Limits

The initial implementation does not support a “pass-thru” DLNARenderer, DLNAServer, or OpenHome Server

That is to say, that the initial implementation of the DLNA/OpenHome servers works strictly off the LocalLibrary and LocalRenderer.

So, a control point that connects to us ALWAYS CHANGES THE SELECTED Library/Renderer to the Local versions, and uses those.

Later, pass-thru versions of these \*could\* be implemented. For example, if the server machine has selected an actual remote device.DLNARenderer device, and a control point asks the server machine to play a song, the service.DLNAServer would pass the request off to the remote device.

This could result in loops.

MACHINE A: Select Machine B as a device.DLNARenderer

MACHINE B: Select Machine A as a device.DLNARenderer

Ask either machine to play a song, and they start sending requests back and forth.

## Coordination of Devices and Services

There are combinations that make good sense, and combinations that don’t make much sense.

For example, if the machine is asked to select a device.LocalRenderer, then the corresponding device.LocalVolume should be set in Artisan.

In fact, a Renderer “device” corresponds to a a DLNA Media Renderer which generally presents two services, where the AVTransport corresponds to my current notion of a Renderer, and the RenderingControl corresponds to my notion of a Volume object, and these two things should always be associated with one another.

In other words, the device.LocalVolume should present itself as a device.service.RenderingControl, and the current renderer should be a device.service.AVTransport, and there should be a new “device” corresponding to the dlna MediaRenderer.

The LocalRenderer corresponds to an dlna AVTransportDevice and the Volume corresponds to a RendererControl device.

Because the expected action is that you are controlling the volume of the device you are attached to.