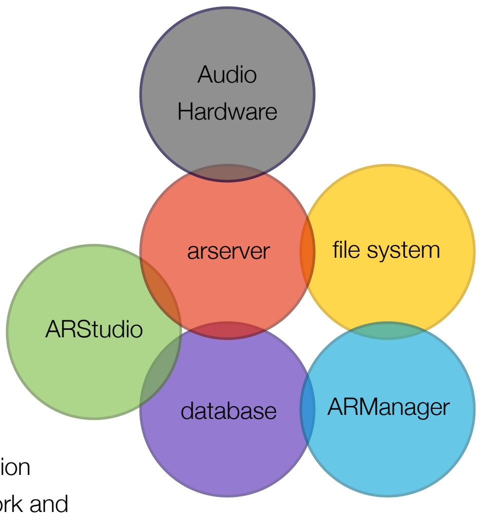
AudioRack Suite

User Guide (Version 3.4.1)

Table of Contents

1.0 Introduction



Component Approach

The AudioRack Suite is modular computer base radio broadcast automation, live assist, and console system. The modularized approach of the system allows it be scaled from the simplest installation with all six components running a single station from a single computer, to a multi-station configuration with components spread across many computers on a network and sharing a common audio library. Before you begin, it is beneficial to understand what these components are and how they interact with each other.

Components and their relationships

Component	Function	Interacts with: Relationship with component
ARServer	Multi-client audio engine: mixer, file player, file recorder, stream encoder, VoIP phone system, and audio hardware host	database: provides automation schedule, logs, audio library file system: provides audio files for playing and recording audio hardware: provides physical audio inputs/outputs Internet: provides connectivity to shoutcast servers
database	Central storage of audio library properties, automation schedule, program logs, etc.	ARServer: accesses automation schedule, logs, audio library ARStudio: accesses audio library for browsing ARManager: accesses user interface/front end
file system	Audio file storage system (local disk drives, network file servers, etc)	ARServer: accesses for audio files for playing and recording ARManager: accesses adding files to the audio library
ARManager	Audio library and automation management user interface	database: provides database back end file system: provides files for adding to the audio library
ARStudio	DJ user interface	ARServer: provides mixer back end database: provides browsing of audio library
audio hardware	Audio inputs and outputs for live audio, monitoring, cueing, etc. Provides a master sample rate clock.	ARServer: Provides audio I/O to the ARServer mixer. At least one audio device is required to provide a master sample rate clock to the mixer

Table 1

1.1 ARServer Component

ARServer is the core component of the AudioRack Suite system. It is responsible for the handling of all the audio associated with the system. ARServer provides a multi-bus (four stereo busses by default) mixing system with ten simultaneous stereo input sources (eight visible to the user interface) by default. Sources can include file players, live inputs tied to audio hardware, and Inter-Asterisk-Exchange (IAX) based Voice Over Internet Protocol (VoIP) channels. ARServer is also responsible for playlist management: mixer input sources are loaded, started, stopped and unloaded under program control to play through a playlist queue. When configured for unattended automation, ARServer will interact with other system components to keep it's playlist queue full by the use of automation rules provided to it. ARServer can also record/encode audio in real time from any of it's mix busses to an output file or a shoutcast streaming server. Each of the mix busses can be routed to any number of audio output hardware devices as well. A 10 second (maximum) broadcast delay is also made available to all outputs and recorders. The mixing system includes advanced monitor mix-minus functions, four broadcast console stye output mute busses, and automatic mix-minus for the IAX channel return feed. A Network Machine Control protocol associated with live audio inputs is also available for controlling external hardware connected to the computer's local network. If and when audio hardware manufacturers begin to support direct monitoring on their MacOS X hardware drivers, ARServer will take advantage of hardware direct monitoring to provide zero latency monitoring with tracking play-through volume control. As of version 1.9.2, ARServer supports the addition of a chain of up to eight AudioUnit effects processors to each player/input source, output group and recorder, processor speed allowing. These effects processors allows for custom equalization and dynamics processing. ARServer also handles logging, track posting, and event triggering of both internal and external tasks.

ARServer is a faceless background application which may be controlled entirely through a simple text based, command/reply style TCP control session. The control interface to ARServer can be accessed by multiple clients (20 by default) simultaneously. A client can be as simple as a telnet session into the control port (9550 by default), or as complex as the ARStudio graphical user interface application, which uses the same control port under the hood.

1.2 database Component

While the database component is not included as part of the AudioRack Suite package, it is most certainly a key part of the system. AudioRack Suite has been design to allow it to work with a multi-user database engine of the user's choosing. To this end, components that interact with the database do so using the libdbi database abstraction library. This allows ARSuite to theoretically use any database server that has a libdbi driver. Unfortunately, subtle differences in how various databases handle queries and built-in functions has made this a bit more difficult in practice. While it is hoped to eventually make the system work with the (free/open source) MySQL, PostgreSQL, and the simple, disk based SQLite3 database systems, only MySQL is currently supported.

The database system, is a central repository for the audio library directory, automation schedules and properties, logs, and metadata for audio files. Like ARServer, the database is expected to be accessible from multiple clients simultaneously. For example, ARServer could be accessing the database making a new log entry, picking a new automation filler, while at the same time a DJ is browsing the audio library for a request with ARStudio, the music director is adding a batch of new music to the library using ARManager, the traffic director is adding a news insert into the schedule using ARManager on another computer, and a listener is using a web based php query to discover what was played two songs ago.

1.3 file system Component

While the file system may not seem like an AudioRack component at first, in a complex AudioRack configuration with components distributed across many computers on a network, the file system which contains the actual audio files

(.mp3, .aif, .m4a) that ARServer plays. This can be very different that just a disk connected to the ARServer computer, especially because some clients may access the audio files on local disks while others may access those same files via a network file server. Therefore, the database has been structured to accommodate diverse volume mounting methods. The database uses a combination of volume information, file Hash UIDs (Unique Identifier), and a user settable disk prefix mounting list, along with tradition file path information to specify a file location. With proper configuration, files can be added to the database when AFP (Apple File Protocol) or other file sharing protocols has been used to access the disk containing the files that the user has mounted (via Connect To Server in the finder). On a different computer, ARServer may access the same files from a statically mounted NFS (unix Network File SYstem) share. In order to facilitate networked file servers, ARServer maintains a 10 second audio buffer for all files being played. This minimizes the risk of a playback drop out due to network delays.

1.4 ARManager Component

This is one of two user interface applications. ARManager provide a front end (either on the same host or through a network) to the database for managing the audio library:

- Adding, correcting and removing files from the library, setting file item metadata such as segue points, artist, album, category, volume and custom information
- Managing the automation schedule and creating automation tasks and automation rotations
- Accessing and analyzing the play history logs
- Building and managing playlists
- Administrative functions such as backing up, merging, and archiving a library, and keeping file references in sync with their disk locations.
- Custom database queries for things that have not been though of yet

This component has a main window for each library that is opened (can open multiple libraries), and additional windows for each item instance in that library that is being edited or viewed. Both kinds of windows are divided into function categories using tab view to save screen real-estate with certain functions (like audio file importing on the main windows) accessed via drop down panels.

This component does not interact with ARServer directly. When managing files, items can be played/previewed from with in the application using the system sound mechanism and quicktime as long as the file system which contains the file is available. None of the advanced audio features which are part of ARServer are available to ARManager.

1.5 ARStudio Component

This is the other user interface application. ARStudio provides a front end (again, either on the same host or through a network) to the ARServer component. ARStudio provides a graphical and optional MIDI control surface interface to the ARServer mixer as well as a graphical interface to the recording/encoding system, play list queue, and the audio library for browsing purposes. Access to the broadcast delay and script viewing are also provided. A graphical interface is also provided to configure most of the major ARServer functions: audio hardware, VoIP, database connectivity, etc. Configuration settings and advanced functions not available through the graphical interface can be access via a console text interface to the ARServer control session. Access to the console provides an opportunity to watch and learn the underlying communications protocol between this client and the ARServer component. ARStudio does not handle any audio itself: all cueing and playing a performed by ARServer.

The user interface is broken down into individual windows for various function: The three main windows:Play List Queue, Library Browser, and Mixer; And various lesser used windows: Script, Logs, Delay, Console, and ARServer log viewer. In addition, the Preference window is available for ARStudio configuration of the visual appearance, ARServer connection properties, MIDI control surface settings and via a drop-down panel, ARServer configuration. The ARServer configuration is perform in a drop-down panel to set it apart for the main Preferences window who's setting apply to the ARStudio directly. The ARServer settings are applied to the ARServer application with which ARStudio is communicating.

1.6 audio hardware Component

Audio hardware is obviously important if you want to be able to monitor your mix, cue tracks, feed live audio into your mix and send your mix to a transmitter, external recorder, etc. Audio hardware interacts with the mixer inside of ARServer, and in fact ARServer requires at a minimum one audio hardware output device to provide it with a master digital audio sample clock. In the simplest case this can be the built-in audio hardware on the computer running ARServer. Even if you are using AudioRack Suite to encode a shoutcast stream for the internet and you have no interest in analog audio outputs or inputs, you will still need to configure ARServer to use one audio device so ARServer can make use of the device's clock. You could for example, configure such a device as a destination for the cue or alternate mix bus that you may not be using in such a configuration any way. Or you could set that devices volume (with in ARServer) to zero forcing it to be silent, though it still provides a clock. The ARServer mixer runs at a 96,000 sample per second clock (by default). It will perform any sample rate conversion needed to interface it's internal sample rate with each of it's audio hardware devices, including the master device from which it is deriving it's sample clock. It will also handle clock synchronization required to interface with devices that are not locked to the same clock as the master device. The only requirement for AudioRack Suite to use a device is that the device have MacOS X CoreAudio drivers and that the device is connected to the computer running the ARServer component. Multiple instances of ARServer can share a device as well.

2.0 Installation Examples

Basic Requirements

The AudioRack Suite installer package will install the ARServer, ARStudio and ARManager components on a computer running Mac OS 10.6 or later. In the simplest configuration, ARServer can use it's host machine's built-in sound hardware and file system for those required components. In a more complex system, you can configure ARServer to use networked disks and Firewire, USB or PCI base sound hardware. Although access to a database server is not required to run ARServer and ARStudio, functionality with out it will be minimal: no automation, logs, audio library, etc. If you require these functions, you will have to set up a database server - highly recommended. AudioRack Suite can create and format a new database on the server for you, but it can not configure the server it self. From this point forward, this guide will assume you have access to the database component.

Currently AudioRack Suite only supports the MySQL 5.1 and latter database server. Goto http://mysql.org to download the free MySQL server and learn how to configure it if you do not already have access to a MySQL server.

2.1 Single Station

In single station mode, there can only be one ARServer instance running on a host computer. If you need to have one computer host more than one audio feed (station) at a time, you can proceed to section 2.2 to learn how to run multiple concurrent instances of ARServer on the same computer. However, it is highly recommended that you read through this section first if you are not yet familiar with AudioRack Suite.

2.1.1 Single Computer

This is the simplest configuration of AudioRack Suite in which all components are hosted on a single computer, with only one instance if of the ARServer component running at a time. This is a good place to get started with AudioRack Suite, as you can migrate from this configuration to to a more complex system latter. For simplicity sake, and in keeping with the idea of having all AudioRack Suite components on the same computer, this section will assume you have a MySQL server installed on the target machine. See later section of this chapter if you want to spread the components across multiple computers.

You will be doing the following:

1. Install The AudioRack Suite software on a single target computer.
2. Configure ARServer and the audio hardware using the ARStudio application so you can play audio files from the target computer's local disk drives and play live audio inputs.
3. Create an new audio library, and add some test audio files and automation rules using the ARManager application.
4. Configure ARServer to use the new library and run automation.
5. Optionally, configure ARServer to automatically run at system startup.
6. Optionally, configure ARServer to automatically start a shoutcast stream encoder and perform other startup functions when it starts running.

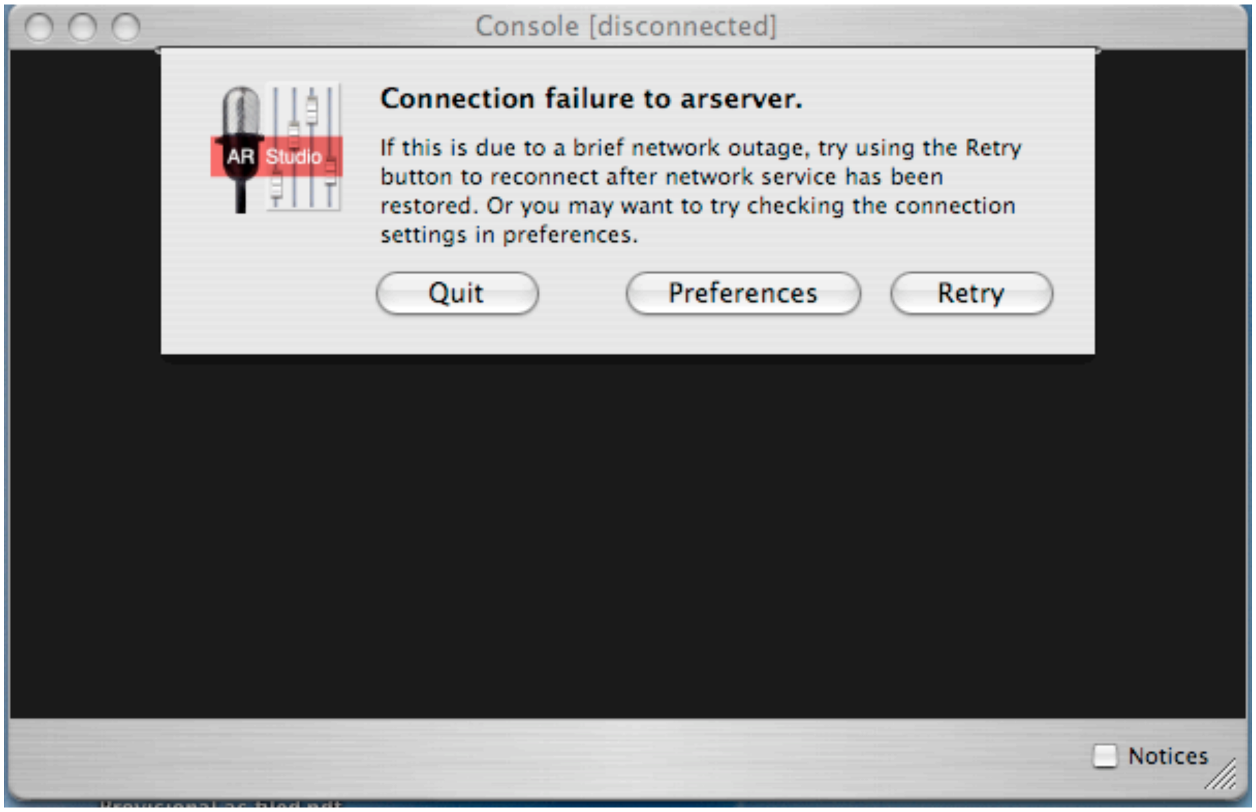
Please have the following ready before you proceed:

1. **A working MySQL database server on the target machine.**
2. **Username and password to access the MySQL server with "create," "insert," and "update" privileges.**
3. **Username and password for an administrator privileged account on the target computer.**

1. Install the AudioRack Suite package on the target computer.
Run the installer. You will be asked to agree to the license and read a description of what the installer will do. If you are upgrading from an older version of AudioRack suite, pay particular attention to any upgrade instructions before you continue. Next you will be asked to select a disk to install AudioRack Suite onto - this is usually your system/startup disk.

Finally, You will be prompted for an administrator username and password. This is required to install some system level components such as libdbi, and auto-startup scripts and create a “audiorack” user on the system which many AudioRack components will use.

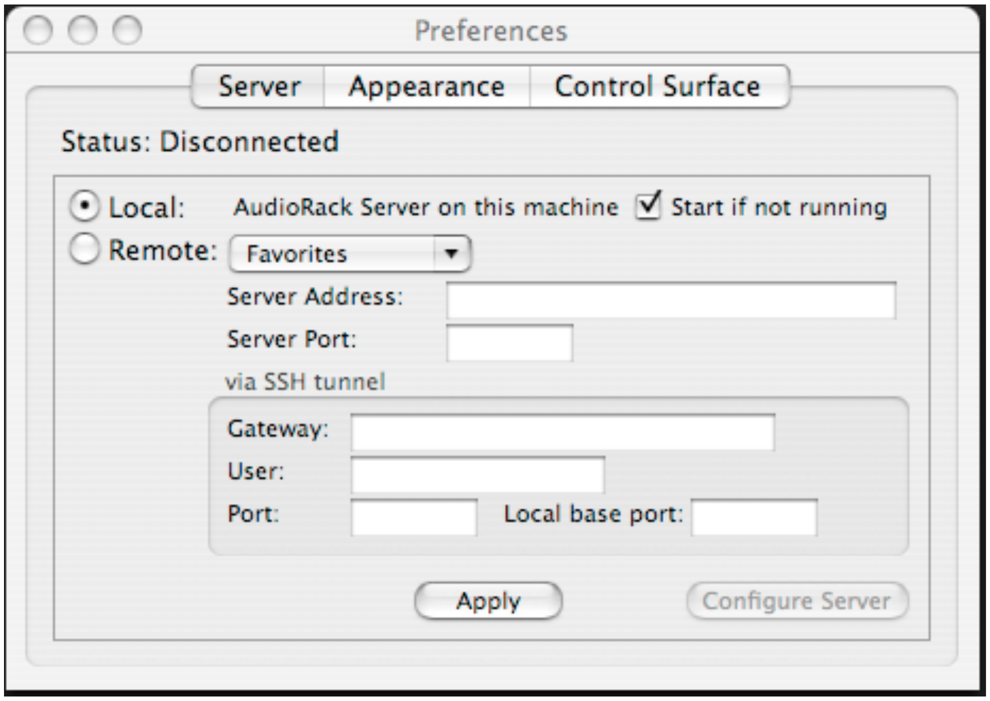
2. Configure ARServer using the ARStudio application.
When the installer finishes, run the newly installed **ARStudio** application inside the **ARSuite** folder with in the **Applications** folder of your system/startup disk. The following window will appear when you run ARStudio for the first time:



Connection failure to arserver.

If this is due to a brief network outage, try using the Retry button to reconnect after network service has been restored. Or you may want to try checking the connection settings in preferences.

Click on the middle **Preferences** button to begin configuration process. The following window will appear:



Status: Disconnected

● Local: AudioRack Server on this machine ✓ Start if not running

○ Remote: Favorites

Server Address:

Server Port:

via SSH tunnel

Gateway:

User:

Port: Local base port:

Make sure the **Local** button is selected and the **Start if not running** check box is checked. These settings tell ARStudio that it will be controlling an ARServer instance running on the same computer as it is, and that ARStudio should start ARServer running if it isn't already running when it tries to connect to it. Note that when ARStudio starts ARServer running, ARServer will be running in the same system user account that ARStudio was started from. All subsequent preferences and setting will be stored in that users account and will not apply if a different user causes ARServer to run. Also note that in this mode, only one user can be running ARServer at a time, and switching users out of the account that started it will cause ARServer's audio to mute as would be expected for any OS X application. latter, in step 5, you will learn how to make ARServer automatically run in a special background "arserver" user account at system startup time.

Now click the **Apply** button. The Console window will appear and show ARServer starting up. Click back on the Preferences window. Once ARServer has started, the Status indication on the top left will change from Disconnected to Connected, and the **Configure Server** button will un-dim.

Click the **Configure Server** button to open the ARServer settings panel. Once open, click on the **Outputs** tab to configure the audio output hardware. Click the **New** button under the **Output Groups** list. This will create and select a new output group. In the Group Configuration area to the right, choose an Audio Device from the list. This list shows all the audio output devices connected to the computer on which ARServer is running. At a minimum, the list should show **Built-in Audio**. Next check the **Master Output** check box. This tells ARServer to use this output group's device as the master clock for it's mixer. Only one output group can be assigned as a master output. If you create more output groups, and assign them as master as well, the last group you assigned as master will be the master output. Note: in order for the ARServer mixer to run, there must be one output group configured as master.

By default, ARServer has four stereo mix busses. In the Channel Mapping area, you assign any channels from the mix busses (shown on the top) to output channels on the audio devices hardware (shown on the left). For example, if you have selected Built-in Audio for the audio device, there will most likely be two channels show on the left. By checking the check box that where channel 1 on the left crosses Mon L on the top, you would be assigning the Built-In Audio output devices channel 1 output to be feed from the Mixers Monitor Left channel. Likewise, you would most likely want to assign channel 2 to the Mon R bus. All channels do NOT need to be assigned.   For now, ignore all the other settings and enter an new name like “MonitorGroup” with no spaces in the **Rename** field. Click the **Save** button, and MonitorGroup will appear in the list at the left. If you have additional audio hardware you want to configure as well, simply create additional Output Groups configured to those devices. You can have multiple output groups that use the same device. For example, you could create a CueGroup that also uses the Built-in Audio device, and assign channel 1 and channel 2 of that device to the Cue L and Cue R mix bus. This will place both the Monitor and the Cue mix busses on channel 1 and 2 of the Built-in Audio Device.

Next, click on the **Inputs** tab to configure the audio input hardware. This panel is very similar to the output panel. The primary difference being that channel mapping is between device inputs shown on the left, and left and right channel inputs to the ARServer mixer. Note also that Input Groups are not actively connected to the mixer until the user “loads” an input. We are simply defining an input group here so that ARServer knows what to do when a user calls upon it to actually start using the input group.

Click the **New** button under the **Input Groups** list. This will create and select a new input group. In the Group Configuration area to the right, choose an Audio Device from the list. This list shows all the audio input devices connected to the computer on which ARServer is running. At a minimum, the list should show **Built-in Audio**, unless your computer doesn’t have a built-in audio input.

As was done with output groups, the Channel Mapping area lets you map which device input channels (shown on the left) you want assigned to the group’s left and right feed to the mixer. For example, if you have selected Built-in Audio for the audio device, there will most likely be two channels show on the left. By checking the check box that where channel 1 on the left crosses the column labeled Left on the top, you would be assigning the Built-In Audio output devices channel 1 input to be feed to the Mixers Left channel input. Likewise, you would most likely want to assign channel 2 to the Right input. All channels do NOT need to be assigned. And if you plan to have a single channel (mono) source such as a microphone, you can assign a device input channel to BOTH the left and right mixer inputs.

Next, in the Default Bus area, check the **Monitor** and **Main** check boxes. This tells ARServer that when it starts playing from this input group, the group’s left and right channels should be assigned to both the Main and Monitor mix busses; this is the same as channel assignment on a mutli-bus analog mixer, or the assignment button on a broadcast console.

For now, ignore all the other settings and enter an new name like “LiveMic” with no spaces in the **Rename** field. Click the **Save** button, and LiveMic will appear in the list at the left. If you have additional audio hardware you want to configure as well, simply create additional Input Groups configured to those devices.

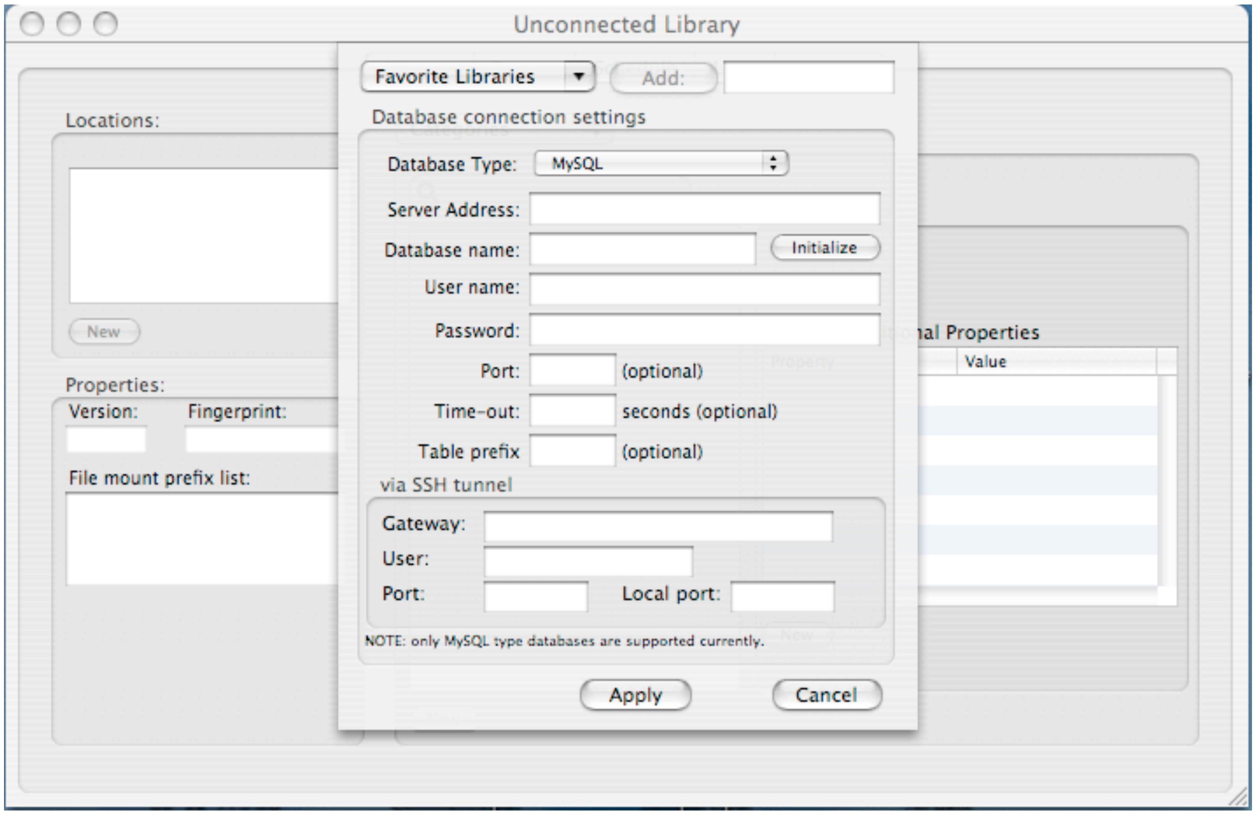
The final thing to do in the ARServer Settings Panel is to make sure **auto-shutdown mode** is checked under the **Special Modes** area of the **General** setting tab. This will make sure ARServer quits when no more clients are connected... when ARStudio quits, if no one else is connected to ARServer, it will quit too.

You are now ready to playing some audio files. Click the **Close** button at the bottom of the ARServer settings panel and close the Preferences window. If the Mixer window is not visible, got to the Window menu and select Mixer to make it visible. Switch to the Finder and locate an audio file you would like to play. Drag the file into the space in the Mixer window below the Players heading. This is the mixer source area. When you drop a file into this area, the file becomes an mixer input channel at the position you drop it with play and stop buttons, volume, pan controls, and count up and down timers. The top left of the channel has a tiny eject button, which unloads the file player from the mixer. Press the play button to start the file playing. Press the stop button to pause it. A jog slider just under the time counter allows you to move the play position. You can Load more than one file into different mixer source channels and manually unload them when you are finished.

Finally, try playing a live audio input. If the Browser window is not visible, goto the Window menu and select Browser. Select **Inputs** from the list on the left to show the input groups you previously defines in the next column. Pick an input group from the list and drag it into an open space in the Mixer window player area. That spot should now load an input group player similar to the file player. Again, the play button turns the input group on. Stop will mute the input. Note that input groups do not normally have a countdown timer. It is possible to set a duration for a loaded input group but that will not be covered here. As with the file player, used the eject button to unload the input group player.

When you are done playing, you can quit ARStudio as we will now turn our attention to the audio library.

3. Create a new audio library, add some audio file and automation rules to it
Run the newly installed **ARManager** application inside the **ARSuite** folder with in the **Applications** folder of your system/startup disk. The following window will appear:



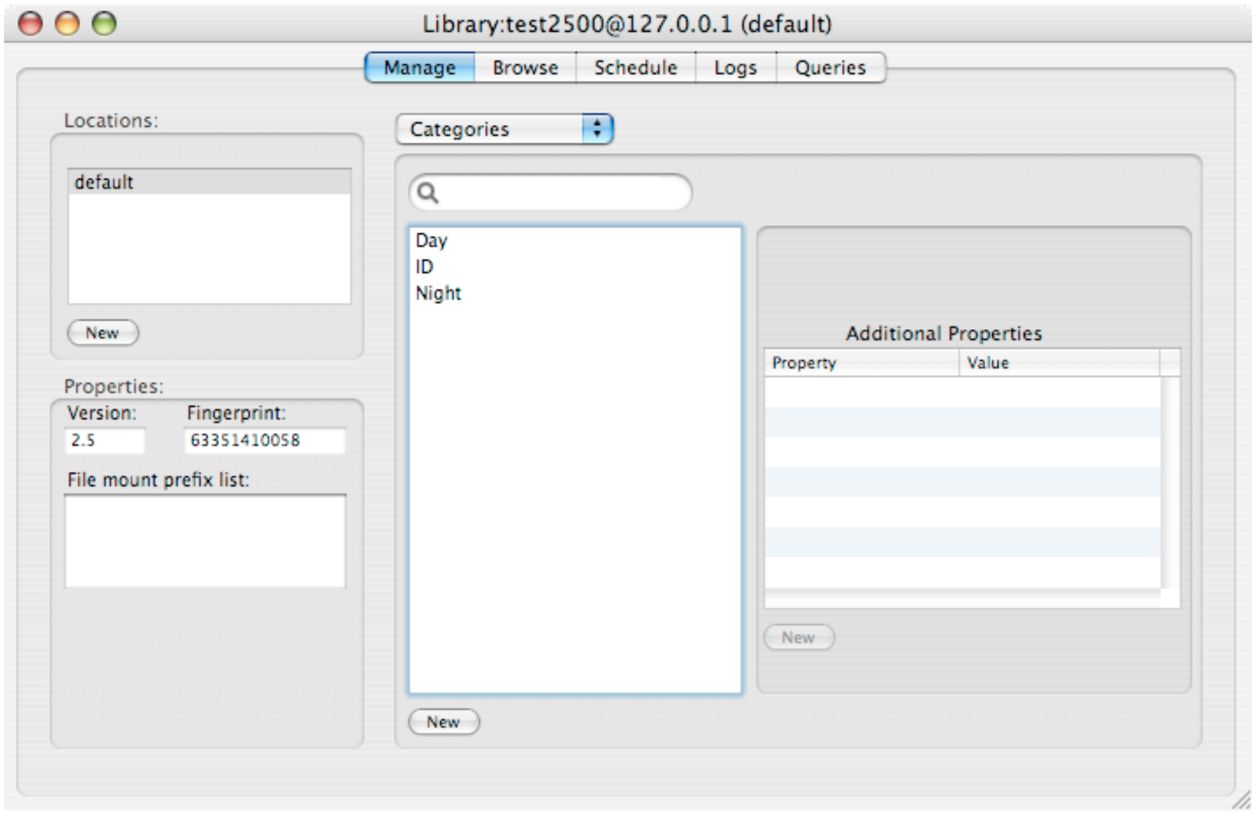
Enter the database server address, username and password. Since the database server is running on this machine, the Server Address will be **localhost**. You can leave the Time-out field bank for now, and you can leave the port field blank if the MySQL server is running on it's default port. Enter any name you desire in the Database Name field as long as it contain only alphanumeric characters. For example "RadioLibrary". Note that it is not trivial to rename a database on the server, so pick a name that you won't likely change later. If you already have a database on the server you would like to use, enter that name in this field. By default, all the database table names that will be used by AudioRack Suite in the new or existing database will have an "ar_" prefix. This will prevent table name conflicts if you are using or decide to use the same database on the server for other things. If you would like to use some other table name prefix, enter that in the optional Table prefix field.

You are now ready to create (if it doesn't already exist) and initialize the database on the server for AudioRack Suite to use. Click the **Initialize** button next to the database name field.

If errors are reported, note what the error is and check with the database server documentation to see what went wrong. For example, the database username you entered may not have had sufficient privileges to create a new database on the server. Or the password for the specified database user may have been in correct. The database server mis-configuration possibilities are well beyond the scope of this users guide.

If all goes well, and there are no errors reported, everything is ready for AudioRack Suite to start using the database on this server. It is a good idea to save these settings for the next time you run ARManager by entering a name in the field next to the dimmed **Add:** button, at the top of the panel. When you enter a name here, the **Add**: button will un-dim. If you click the button, the current setting on the panel will be saved under the name you entered in the **Favorite Libraries** popup list on the upper left of the panel. This information is actually saved in your user Preferences folder, so it will not be available to other users on the same computer who might run ARManager. The next time you run ARManager, you can select this favorite from the list so you do not need to re-enter the settings again.

Click the **Apply** button at the bottom of the panel. The panel will close leaving the now connected and active library window visible.



Next we will create a new "location" with in the library. The ARServer component of AudioRack is responsible for actually playing audio files, logging what has been played, and performing automation and playlist functions. We partially configuring this component already, but we did not hook it into a database, so all the database related functions are not available yet from ARServer. We will be configuring ARServer to use the new library you just created shortly, but first you will want to create a new "location" with in the library for ARServer to put it's logs into, and to look for it's automation schedule. An AudioRack library always contains a “default” location, which is selected by default when you open a library in ARManager. In this simplified configuration of AudioRack Suite, there is only one station using the library. However, it is possible in more complex configurations for many stations, each with their own logs, automation properties, etc. to share a common library. This is achieved by creating multiple locations with in the same library. We will create a new location now, just for good measure.

Make sure the **Manage** tab is selected at the top of the window. Next, click the **New** button with in the **locations** area on the top left of the library window. The locations list will now show **default** and **New Location** with New Location highlighted. Type a new name for the location, for example **MyRadioStation**, and hit the enter key. It is trivial to rename the location later, as long as you remember to change the name in the ARServer settings (to be covered shortly) as well.

Next we will add some audio files to the library. Although AudioRack Suite can accept files from any storage location that is accessible to both ARManager (when you add the file) and ARServer (when the files are actually played), it is convenient to keep all you audio file in rooted in a a common location, be it a special disk, or a particular directory. Although you are not required to have all you audio in such a location, for the sake of simplifying instructions, this guide will assume all the audio files we are about to add are rooted in one directory. Rooted, in this case, means that files of interest can reside with in any number of sub directories with in a root directory. The default iTunes “Music” directory if s good example of such a scheme: actual audio files are stored in album named subdirectories, all with in a main directory called “Music” in your home folder. If you have files scattered in numerous directories, you will need to repeat the process which follows for each root directory which contains audio files you want to add to the library.

Migrating to a multi-computer system:

It will be much easier to migrate from a single computer AudioRack Suite system to a multi-computer system if all your audio files are on one or more disks or partitions OTHER THAN YOUR SYSTEM DISK. Using other disks/partitions will enable you to share those disks/partitions with other computers as distinct disk volumes, such that they appear to both the host system and the shared systems as distinct disks from the computers’ system disk. The system disk always appears as “/” in the file hierarchy to the host system only, rather than appearing at any of the standard system mount points such as “/Volumes/” We will want all the audio files to be available via the standard mount points to ALL THE COMPUTERS in the AudioRack system if you go to a multi-computer AudioRack system.

With a new Library window selected, go to the **File** menu and select the **Import** menu item. A submenu list will appear to the right. Select the **Audio Files** submenu from the list. A File Import Panel will drop down over the Library window. You can now drag the files you want to import or the root folder containing the files from the finder into the list area at the top of the panel, or click on the **Add** button to use a standard file browsing dialog box to navigate to files or folders of interest. You can add as many files or folders as you like. When you add/drag a folder, the list will traverse the folder and all it’s subfolders adding all the files found with in to the list of files to be imported. You can select a file (or multiple files using the shift or control key) from the file list and delete (using the delete key or the **Delete** menu item under the **Edit** menu) files you are not interested in adding.

Just below the file list, there is a pop-up menu labeled “import into category.” By selecting a category from this list other than **[skip category]** you can make ARManager add the files in the file list to a particular category at the same time it added them to the library. However, this is a new library, so there are no categories. We will take a set back now and create a new category to add all the audio file you have selected into.

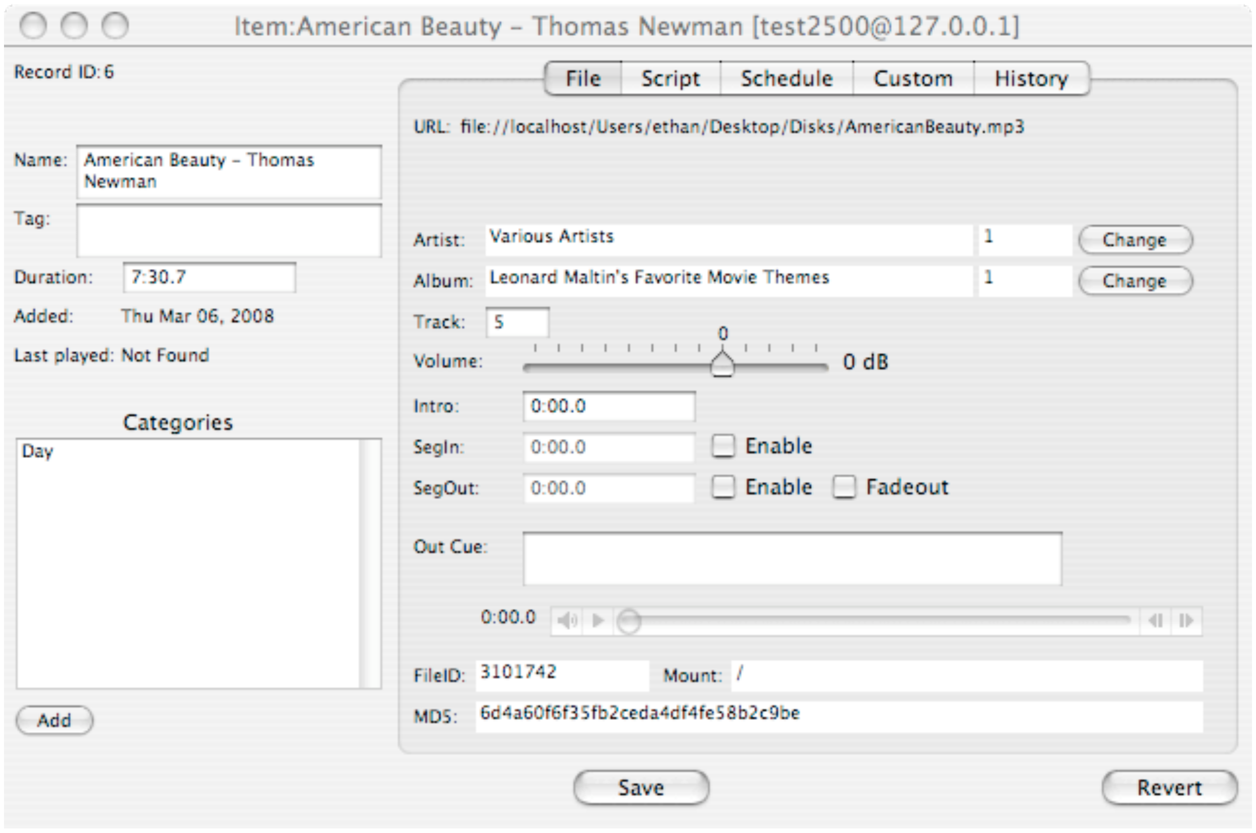
Categories provide a convenient method to browse through the audio collection in the library and provided the simplest method for setting up automation. We will create a new category called “Test Category” in this example. You will probably want to create various meaningful category names like Classic Jazz, Daytime Promo, or something to your liking to classify you audio material latter. You can add and remove files from categories, and rename categories any time latter as well.

Click the **Close** button to dismiss the import window. The file list will not be lost... when you reopen the import panel, the setting will remain as you have left them. Make sure the **Manage** tab is selected at the top of the window. Near the top of the window there is a pop-up menu button with a list containing “Category,” “Artist” and “Album” entries. Make sure **Category** is selected, then click the **New** button in the area below the pop-up menu. This will create and select a new category in the category list. Type the name “Night” for the new category and hit the return/enter key. You have just created a new category called “Night” Repeat the process one more time to create another new category called “ID” and again for one called “Day.” Go back to the audio file import panel (again, under the File->import->Audio Files menu), and select the newly created “Day” category from the “import into category” pop-up menu.

At the bottom of the audio files import panel you will see a “File is considered duplicate if...” area and a “Duplicate item handing” area. These areas are for settings regarding what ARManager should consider a duplicate file and what action to take when it tries to import a duplicate file. For now, leave these as they are with there default settings.

Finally, click the **Import** button and ARManager will go through the files in the list trying to first import them into the library, noting item duration, file location properties, and creating Artist and Album entries as required from the audio meta data tags if any. Then, if added, the new item is added to the category “Day” As are Manager progresses, the status of each item is noted in the list: Added, Skipped (not an audio file or skipped if it’s a duplicate item and ignore was selected for duplicate items), Updated (if it’s a duplicate item and update category was selected for duplicate items) or File (if it’s a duplicate item and update file location was selected for duplicate items). You can now close the import window.

You can now browse the audio in your library by clicking the **Browse** tab at the top of the Library window. A column style browser will appear. Select a browsing method in the left column (such as Artist) and the next column to the right will list all the artists currently in the library. Clicking on an artist in the list will bring up more options in the next column and so on until you get down to a single item. If you select **Category** in the left most column you should see the three new categories we created. Clicking on the **Day** category will make all the items we imported available as they were all put in this category as they were imported. The other two categories will be empty. Once you have browsed down to an item, you can click the **Open Item** menu item in the **File** menu to open this specific item in an item editor window for viewing and editing this items specific properties.



Record ID: 6

Name: American Beauty - Thomas Newman

Tag:

Duration: 7:30.7

Added: Thu Mar 06, 2008

Last played: Not Found

Categories

Day

URL: file://localhost/Users/ethan/Desktop/Disks/AmericanBeauty.mp3

Artist: Various Artists 1

Album: Leonard Maltin's Favorite Movie Themes 1

Track: 5

Volume: 0 dB

Intro: 0:00.0

SegIn: 0:00.0

SegOut: 0:00.0

Out Cue:

0:00.0

FileID: 3101742 Mount: /

MD5: 6d4a60f6f35fb2ceda4df4fe58b2c9be

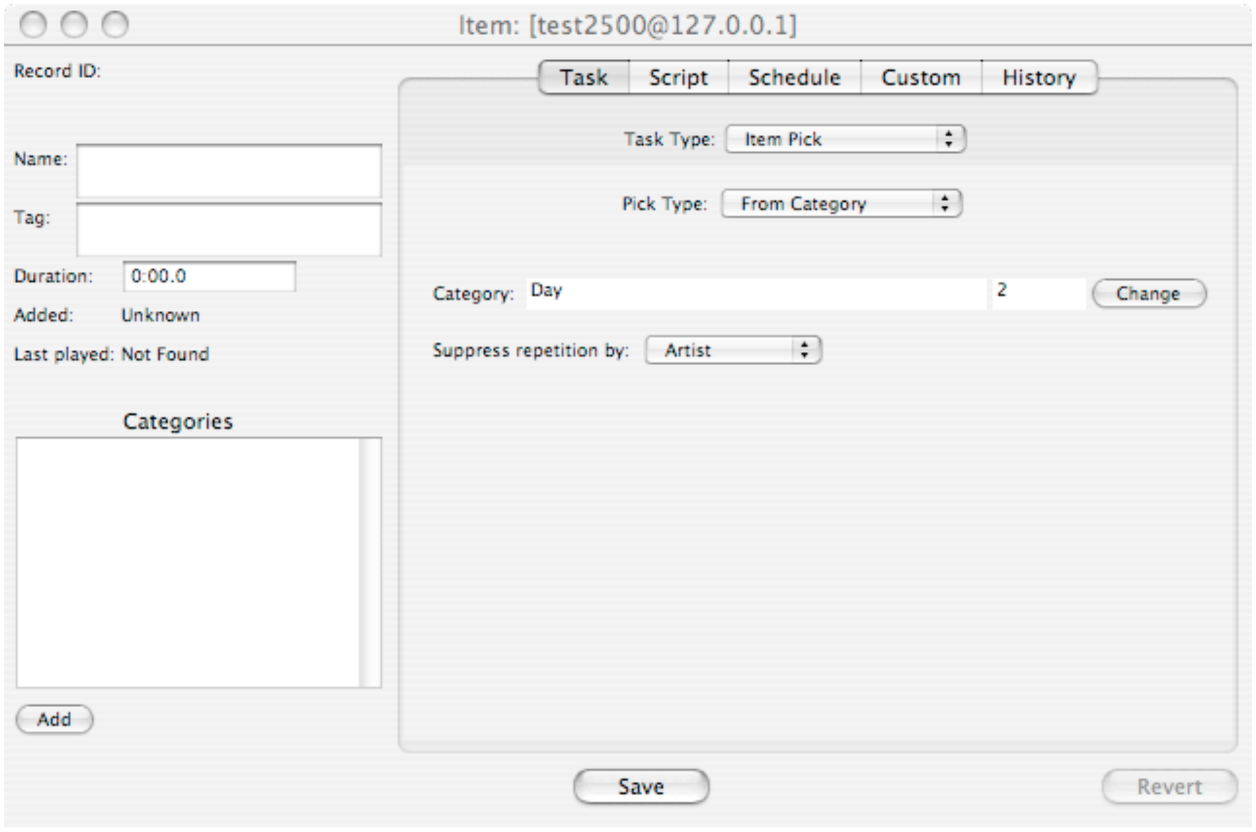
At this point, you will want to add some items to the other categories so we can demonstrate how to set up automation. To do this, simply click the **Add** button under the **Categories** list in the Item window. You will notice that all the items you added to the library are already in the Day category. After you click the Add button, a panel will appear listing all the categories available which the item is NOT ALREADY in. Select an addition category, either Night, or ID and click OK to add the item to that category. Repeat if you want to put the item in more categories. You can click on a category the item is currently in from the Categories list in the item window and hit the delete key to remove it from that category. When you are done setting categories for the item, click the **Save** button to apply the changes to the database. Close the Item window and proceed to pick another item from the Library browser. Add this item to some other categories too, and continue until there are at least a few items in each category.

To finish our work in the library for this demonstration, we will set up some automation rules. Back at the Library window, make sure the **Manage** tab is selected at the top of the window. In the locations list near the top left, select the **MyRadioStation** location. This will tell ARManager that any schedule properties we set up now, will apply only to the MyRadioStation location. Next, in the **File** menu, selecting the **New Item** menu item and the **Task** submenu item. A blank Task Item window will appear, similar to the file item windows you had previously worked with. We will create the first of three task here that will facilitate automation. But first some background information. AudioRack supports three types of library items: File Items, Task Items, and Playlist Items. File items behave as you might expect: You can put a file item in a ARStudio player, or in the ARStudio play list Queue. ARServer will then play the file item when called upon to do so. Task items and Playlist Items can only be added to the play list queue. A Playlist item is an ordered list of items of any kind. When the play list queue gets down to a playlist item (a.k.a. the playlist item is next to be played), it expands the item into it's list contents.

A Task items behavior in the queue depends on what type of task it is. Some tasks, like the Item Pick task, will run immediately when added to the queue. An Item Pick Task is designed to perform some database queries which result in the Task itself being replaced in the queue by the item/items that result from the query. For example, a Task might perform a category pick from the Day category. When this task is added to the queue, it will run and pick, according to some rules, an item from the library in the Day category. Once an item has been picked, the task will be replaced in the list by the chosen item.

Other tasks, like ARServer Command tasks, Open File Tasks and Shell Command tasks will remain the the queue like a Playlist Item does until the queue gets down to it being next. When it is next, the task will run, and be deleted from the queue when it is finished. These tasks can be used to schedule things to happen on the computer outside of ARServer (Open File or run Shell Commands) or to make ARServer perform internal function or change settings using it's control interface command set.

Now, back to our new task window. We are going to create an Item Pick task to pick items from the Day category.


Item: [test2500@127.0.0.1]

Record ID:

Name:

Tag:

Duration: 0:00.0

Added: Unknown

Last played: Not Found

Task Type: Item Pick

Pick Type: From Category

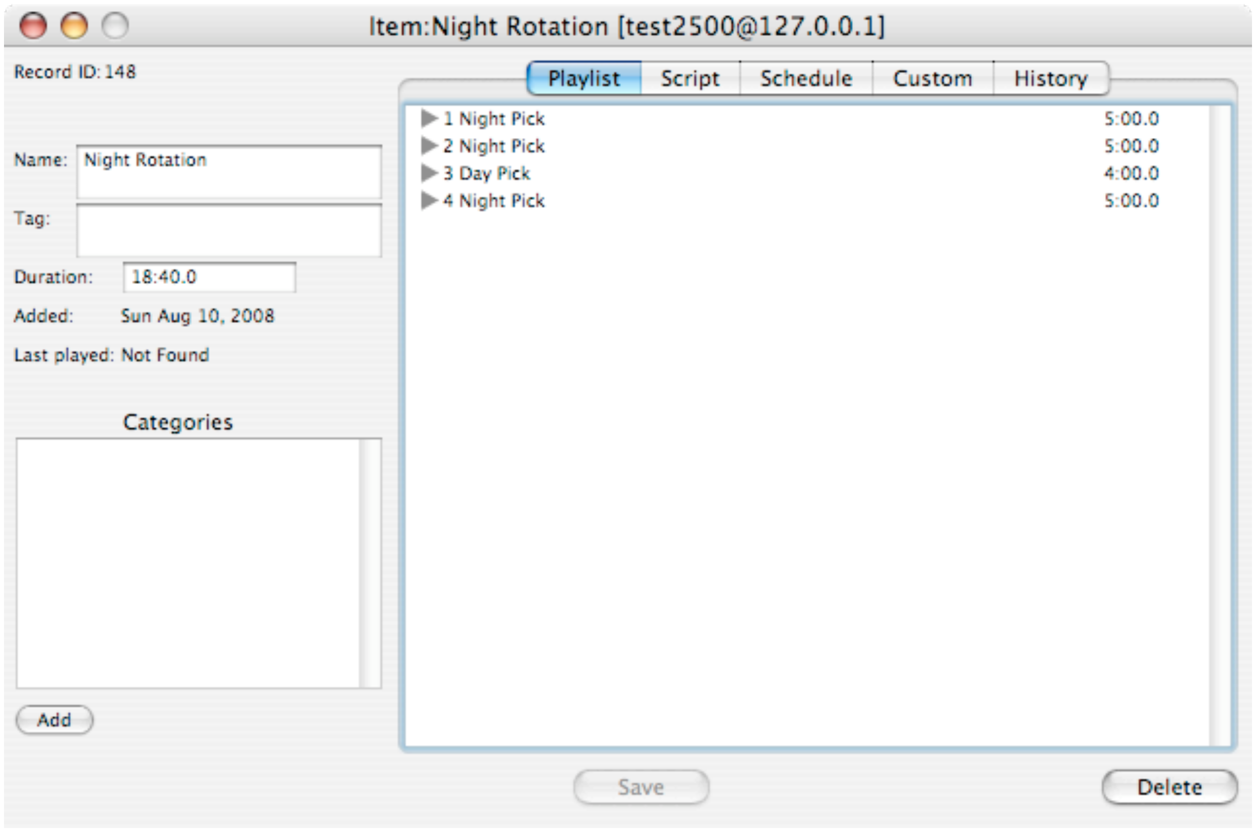
Category: Day 2

Suppress repetition by: Artist

In the Name field at the top left, enter a name like "Day Pick." Next, set the Duration field, just below the Name field to a time which is a good estimate of the duration of the item that this pick will resolve to. For example, if the Day category is mostly pop songs, the a good Duration number would be 4:00 since pop songs are usually around 4 minutes long. This is just an estimate so the automator will have an hint at how to schedule things around the pick before it has been resolved into a final item. Next set the Task Type pop-up menu to **Item Pick** and the Pick Type pop-up menu to **From Category**. The Category field will be blank. Click the **Change** button to the right of the Category field to open the category selection panel and select the Day category from the list. Click the **OK** button to make this the category the task will pick from. The "Suppress repetition by:" pop-up menu now needs to be set. Select **Artist** from the list. What this means is that ARServer, when it runs the task, will query the library for a list of all the items in the Day category and sort the list by the last time all the items Artists were played, most recent last. The list is then cut in half, ensuring the most recent artists will not be chosen again, and a bell curve (normal weighting) random pick is made weighed to prefer items near the front of the list (least recent artist plays). The suppression can be set to item, album or name as well. Name and Item differ in that two items can have the same name, but if they are different files, will always be seen as unique Items. Click the **Save** button at the bottom of the window, then close the window. Repeat the new task procedure to create task to pick from the ID and Night categories as well.

We now have three task items which correspond to each of the three categories in our library. To demonstrate how a Playlist Item can fit into this, we will create a new Playlist Item next.

In the **File** menu, selecting the **New Item** menu item and the **Playlist** submenu item. A blank Playlist Item window will appear.



Record ID: 148

Name: Night Rotation

Tag:

Duration: 18:40.0

Added: Sun Aug 10, 2008

Last played: Not Found

1 Night Pick	5:00.0
2 Night Pick	5:00.0
3 Day Pick	4:00.0
4 Night Pick	5:00.0

Name the Playlist Item Night Rotation. You do not need to set a duration, as the duration will be calculated for you based on the items you put in the play list. Keeping the new Playlist Item window open, switch back to the Library window and select the **Browse** tab. Use the browser to navigate to the Tasks. The three tasks we created should be listed. Drag the Night Pick task to the new Playlist item window and drop it onto the Playlist area. The Night Pick task should appear at the top of the Playlist list. Drag or use copy/paste to add the Night Pick task again in the list. Then drag or copy/paste the Day Pick task from the Browser to the third spot in the Playlist list. Finally, copy/paste or drag the Night Pick task to the last item in the list, and click the **Save** button at the bottom of the new Playlist Item window. You can close this window now.

Finally, we will create an automation schedule. First, we will schedule the ID Pick task. In the Library browser, browse to the ID Pick and open it. In the ID Pick Item window, click on the schedule tab, and click the **New** button. A new schedule entry will appear and be selected. The selected items properties will become editable in the area under the schedule list. Set Day to Any, Month to Every, Date to Every. Set the Hour of the Time field to All and the minutes to 00. Set the Priority to 9 and make sure the Fill box is NOT checked. Click the **Save** button at the bottom of the window and close the window.

What you have just done is scheduled the ID Pick task to get inserted into the ARStudio queue such that it will play at every hour of every day, month, date, etc, at as close to 00 minutes as possible with out interrupting the item playing ahead of it - typical for radio station identification. As the queue plays closer and closer to the item, the item may get shifted around in the queue as the queue attempts to position it as close is it can to it's target time. The priority of nine, a relatively high number on a scale from 1 to 10, will make sure it gets inserted before most other items that may be scheduled for the same time, assuming they have priorities of 8 or less. If the priority is set to the maximum value of 10, then the arsrever will make sure the item plays at EXACTLY the time specified, interrupting the previously playing item as required to start it on time. Because Fill was unchecked, a single ID Pick item will be inserted at the time required.

Next we will schedule the Night Pick. Browse to and open the Night Pick item and go to the items schedule. Again, create a new schedule entry, this time, set the entry to Day to Any, Month to Every, Date to Every. Set the Hour of the Time field to 0 and the minutes to 00. Set the Priority to 5 and make sure the Fill box IS checked. Click the **Save** button at the bottom of the window and close the window.

Like the ID Pick item, we have just scheduled the Night Pick task to be inserted at 0:00 (midnight) every day. Only this time, the Fill check box was checked. What this means is that a Night Pick task will be added repeatedly to the ARStudio queue, starting at midnight, and continuing until something else (of any priority) is scheduled to fill the list. You will see in the next section, that when ARServer is set up to do automation, an automation threshold will be specified. This sets how many items the automator in ARServer will try to keep in the queue at any time. If there are fewer items than the threshold count, the automator will look to the most recently scheduled fill item, relative to the time at the end of the queue, to add more items to the end of the queue until the threshold is met. For automation to work properly, there needs to be at least one item scheduled as a fill at midnight any day. Otherwise at the start of a new day, nothing will play!

At this point, automation will start any time at or after midnight. The automator will keep the queue filled with Night Pick items, which upon insertion in the queue will be run and resolved according to the Night Pick rules we set up into files items from the the Night category. In addition, a single ID Pick will be inserted, run and resolved to play as close to the top of each hour as possible.

Now, repeat the schedule setup process used for the Night Pick item on the Day Pick item, only set the hour to 8 instead of 0. This will modify the automation schedule such that Night Picks will fill the schedule until 8 in the morning at which time Day Picks will fill through the rest of the day until midnight the next day.

To finish the automation set up, create a schedule entry for the Night Rotation Playlist, found in the library browser under playlists. Schedule this item as was done for the Night and Day Pick items, with the hour set to 18 (6 PM in 24 hour time). Now, the automation schedule will behave as before until 6 PM, at which time, the queue will be filled with the Night Rotation for the rest of the evening until midnight the next day. There are some details to note when a playlist is scheduled as a fill item. First, the entire playlist is not added to the queue as you might expect. Instead the automator will step through the playlist adding each item one at a time in sequence until the automation threshold is met. When the queue drops below the automation threshold again, the automator will begin adding to the queue again from the scheduled playlist, starting where it left off last time. Second, when the automator get to the end of the playlist, it will start over again at THE SECOND ITEM IN THE LIST if there is more than one item in the playlist. What this does is provide a mechanism to play an introduction item at the start of a new playlist rotation that will not repeat. Our Night Rotation list was Night Pick, Night Pick, Day Pick, Night Pick. So, staring at 6 PM, the queue will be filled with a Night Pick, then another Night Pick, then a Day Pick, then a Night Pick. Then the playlist will repeat, skipping the first item, so next will be a Night Pick then a Day Pick, then a Night Pick the repeat again skipping the first item and so on until midnight the next day.

You can see a graphical view of how the schedule is set up for any given day by clicking on the Schedule tab on the Library window. Clicking items in this schedule view will select the item so you can open and edit it with out having to find the item in the library browser. Note that items can have more than one entry in there schedules. You could for example schedule ID Picks at half past each hour too, or create three entries for 4:15, 8:30, and 14:43 every day, or maybe just on Fridays, etc.

You are now done with ARManager for the moment. Select Quit from the File menu.

4. Configure ARServer to use the new library and run automation.
Run ARServer again. This time, ARSever will remember the connection setting from the last time you ran it and it will start ARServer and connect to it. Open the Preferences window and click on the **Configure Server** button to show the ARServer configuration panel. Click on the **Automation** tab. Set the Startup state to On so that automation will automatically be turned on when ARServer is started up. Set Queue filling to 8 so that the automator will keep at least 8 items in the queue - this is the automation threshold mentioned earlier. Click the save button second from the bottom of the panel.

Next, click on the Database tab. Enter the same settings as you used to create the Library in ARManager:

Database type will be **MySQL**, Server Address will be **localhost**, Database Name will be **RadioLibrary**, and enter a Username and Password for the database with the same permissions as you used in ARManager.

Click the **Save** button again. The locations list on the left of the panel should show the locations available in the library. Select **MyRadioStation** from the list and click the **Save** button again. You can close the panel, and close the preferences window. You have just finished configuring ARServer to use the new library, the libraries MyRadioStation location will be used for the automation schedule and to store the play logs.

And now, for the moment of truth: Open the Queue window and click the **Auto** button at the top left. Automation will turn start running with the area just below the auto button showing the name of the current fill item. The queue will start filling with Tasks, and tasks will run and resolve into playble items. Once playable items become available, the queue will start loading, starting, stopping and unloading players as it plays through the queue. The Run button/indicator atthe bottom left of the Queue window will show that the queue is running such that when an item in the queue is done playing, the queue will start the next item. The queue will attempt to keep enough items loaded into players such that there is at least one minute of audio cued to follow the currently playing item. Clicking the Break button next to the Run button will prevent the next item from starting when the current item is finished playing. Note that the queue can not be in Break mode while automation is on. Click Live or Off if you want to have a break in the queue.

You can interact with the queue to manipulate items which are not yet playing, selecting them to move or delete. You can also add items by dragging or copy/pasting files from the finder into the queue, again as long as you place the items after items which are currently playing. You can also open the Browser window (similar to the ARManager library browser), to browse the library for items and add them by dragging, copy/pasting, or using the add buttons in the browser window.

5. (Optional) Configure ARServer to automatically run at system startup.
This procedure will set up ARServer so it will automatically run when the computer is started. In this mode, ARServer does not run in a user account, as it did in the previous example, but will be started by the system to run under a special user account named ARServer. Because this users in never really "logged in," you can switch desktop user accounts all day long and ARServer will keep playing. This is contrary to normal application behavior where switching desktop users causes all applications running under switched out users to have their audio muted. Sorry Apple, but this just isn't acceptable to a radio station, so ARServer breaks apples guidelines here.

Because it is not running in the same user account as it was configured in in the previous steps, you will need to reconfigure ARServer after it is running in this new mode. It is also important, since ARServer will now be running shortly after the computer boots, that any disks it may need to play audio are available shortly after the computer starts. This is not a problem for and directly attached disk drives, however, it can be a problem for networked disks. If you are using a network disk for audio, you will need to make sure it is statically mounted. More on that if you choose to do a multi-computer installation. For now, we will assume all the disk are local, in keeping with this being a single computer AUdioRack Suite example. So lets get started.

Important: Make sure all the files in your audio library are at least read accessible from the arserver user account. You may have to play with file permissions, or move your audio files to a different location. Remember that you CAN move files as long as they remain on the same disk they were on when you added them to the Library.

First, quit ARStudio if it is running. Unless you changed the auto-shutdown mode ARServer setting (it should be checked), ARServer will quit along with ARStudio. You can open the Terminal Application and type the following command to see that ARServer has quit:

```
ps -ax | grep arserver<return>
```

The result should be a single line similar to the following:

```
username    7677   2.7  0.0    18052    252  p1  R+    8:10PM   0:00.01 grep arserver
```

This shows that the only process running which contains the name arserver is the process we just ran to is if there were any such processes. If ARServer was still running, you would have seen additional results like this:

```
username    7656  12.3 12.1   187996  79292  ??  Ss    8:04PM   0:42.86 /usr/local/arserver/arserver -k -n
username    7677   2.7  0.0    18052    252  p1  R+    8:10PM   0:00.01 grep arserver
username    7655   0.0  0.0    28032    140  ??  S     8:04PM   0:00.02 /usr/local/arserver/arserver -k
```

If that is the case, and ARServer IS still running, type the following at the Terminal Application prompt:

```
telnet localhost 9550<return>
```

When you see the ARServer prompt ("ars>") type:

```
shutdown<return>
```

You have just accessed the ARServer control port (using the telnet command), and told it to shutdown. Check again to make sure it has stopped, and then you can quit the Terminal Application.

Now that ARServer isn't running, go to the **Applications** folder on your system disk, open the **ARSuite** folder and run the applescript called **ServerStartupEnable.app**. You will be asked for an administrators password as the script runs. When it is finished, ARServer will be running under the asrserver user account, and it will start automatically when ever the system is rebooted.

Note: If you decide to disable ARServer staring at system startup time, run the ServerStartupDisable.app, also in the ARSuite folder. This script will NOT stop a currently running instance of ARServer, it will only prevent it from running automatically again when you reboot the computer.

Now you need to reconfigure the new instance of ARServer since it is running under a new user account. You can do this by running ARStudio and repeating the ARServer setup outlined in steps 2 and steps 4. ARStudio it self will still be running in your user account, so it will remember it's configuration, knowing to connect to a locally running copy of ARServer. You will want to uncheck the **start arserver if not running** option in ARStudio preferences as ARServer will already be running when ever the system is booted.

If you don't want to re-enter all the ARServer settings through the ARStudio interface, and you are familiar with moving files and setting file permissions in the Terminal Application, you can copy the following three files from your home folder into /etc/arserver/

```
~/Library/Preferences/ars_input.conf
~/Library/Preferences/ars_output.conf
~/Library/Preferences/ars_prefs.conf
```

You will need root access to modify the /etc/arserver/ folder. Make certain the owner for all three files after you copy them is set to "root" and the group is set to "arserver" with read and write access for both owner and group. There should be NO access allowed for anyone else for security reasons. These three files are where ARServer stores it's settings. When run in a user account, the settings are stored in ~/Library/Preferences/.

Note: When ARServer runs at startup time under the arserver user account, it is executed by the system with the following shell command:

```
/usr/local/arserver/arserver -c /etc/arserver/ars_startup.conf -k
```

The -c flag specifies and alternate startup configuration file and the -k flag tells it to run in keep-alive mode, so it will restart itself if it should crash for some reason. ARStudio runs it with only the -k flag. With out the -c flag, ARServer will look first in the user preferences for the ars_startup.conf, then look in /usr/local/arsserver for the default file created by the installer if one isn't found in the user account preferences.

6. (Optional) Automatically start a shoutcast stream encoder and other functions.

Shoutcast encoder startup:

The default startup script for ARServer will look for a file called ars_stream.conf to carry out the configuration and starting of a shoutcast stream encoder during the ARServer startup cycle. This file does not exist be default, so you will need to create it. Where this file should be stored depends on where ARServer is starting up from: is it running at system startup time as optionally configured in the previous step or is it running under a user account?

If it is starting in a user account, then the default startup script will direct ARServer to look for this file in the users ~/Library/Preferences/ folder. If arsrever was started at system startup time, then the file needs to be in the /etc/arserver/ folder. Accessing and modifying this folder will require root user access.

To make things a bit easier, an example file has been installed for you to copy and edit. The file is called stream_example.conf and is located in the /usr/local/arserver/ folder. Using the Terminal Application, copy this file into either your user account preferences or into /etc/arsrever/ using the following command at the terminal prompt:

cp /usr/local/arserver/tream_example.conf ~/Library/Preferences/ars_stream.conf

for user account startup, where the Terminal Application is running from that user account

or

sudo cp /usr/local/arserver/tream_example.conf /etc/arserver/ars_stream.conf

for system startup, where Terminal Application is running from an administrator account. You will be asked for an administrator password.

For the user account situation, you can use TextEdit Application to edit the file, or use the text editor of your liking. For the system startup situation, you will need to edit the file using a command line based editor like pico from within the Terminal Application. Type the following command at the terminal prompt:

sudo pico /etc/arserver/ars_stream.conf

This will run the pico editor in the Terminal Application window. When you are done editing the file with pico, hit control-o to save the file then control-x to quit pico.

The example file should be self explanatory. Replace all the text between and including the "< >" characters on each line with values appropriate for you shoutcast stream. You will need to have a shoutcast streaming server already set up and waiting to except your stream. In addition, you can add the following line to the end of the file:

lockrec -1

if you want to lock the encoder to prevent DJ's from making changes to it in the ARStudio user interface. Note that the last line in this file must be followed by a carriage return (new blank line). The first uncommented line creates a new, bland recorder/encoder. The lines that follow set recorder properties, initialize and start the new recorder/encoder. Normally, each of these ARServer commands are followed by the new recorders unique ID code (UID). In this case, specifying -1 for a UID tells ARServer to use the most recently used UID, which is the UID it just created from the newrec command.

If you do want to stop or otherwise change the encoder after having locked it, you will need to get the UID of the recorder using the **rstat** command to list all the recorders and their status and UIDs, and then use the **unlockrec** command followed by the UID of the encoder to unlock. These commands can be issued from the ARStudio Console window, or by telneting into arsrever from the Terminal Application as shown on page 18. See Appendix A for more information on these and other ARServer commands.

After saving the edited file, ARServer will load and run a shoutcast encoder the next time you run it. ARStudio will show the encoder and it's status in the Recorder area of the Mixer window.

Other startup functions:

You can configure ARServer to perform any number of other functions at startup time by editing the ars_stratup.conf file. You can add any ARServer commands found in Appendix A to this script, including references to other script files using the **config** command. An example of a useful startup command would be the loading of a microphone input to specific player. Like editing the ars_stream.conf file, you need to consider where ARServer is run from to make sure you edit the correct ars_startup.conf file.

If ARServer is running at system startup time, then you will need to edit ars_startup.conf found in the /etc/arserver/ folder. The file is already present in this location, so all you need to do is edit it. Again, you need root access to edit this file, as was required when editing ars_stream.conf.

If ARServer is running in a user account you have two options - edit the default ars_startup.conf file found in /usr/local/arserver, or create a local copy of the file in a specific users preferences folder. Editing the default file will change the default behavior of the ARServer startup process for all users who do not have custom ars_startup.conf files in their preferences folder. You will need root access to edit this file just like the file located in the /etc/arserver/ folder. If you just want to change the startup process when ARServer is run by a particular user, then you should copy the ars_startup.conf file from /usr/local/arserver to ~/Library/Preferences/ars_startup.conf, as the user who's account you are copying it into, then edit the copy.

When ARServer is run again, it will startup using the edited ars_startup.conf file.

2.1.2 Multi-computer

It is simple to reconfigure AudioRack Suite from the single station, single computer configuration to use different computers for different components. The only two components which will need to be together on the same machine are ARServer and the audio hardware it will use. The example which follows will demonstrate how to run AudioRack Suite with each component on a different computer. Note that it Is required that all the computers across which AudioRack will be spread must have network access to each other.

To begin, we will assume you have five computers on a local network with IP addresses of 192.168.0.1 through 192.168.0.5. The computers need to be on the same network or sub-net so long as they all have a routes through which they can see each other. We will be running the database server on the 192.168.0.1 compute, running a file server on the 192.168.0.2 computer, running ARServer with the audio hardware on the 192.168.0.3 computer and running ARStudio and ARManager on the 192.168.0.4 and 192.168.0.5 computers. With this configuration, you can run ARManager and/or ARManager on any or all the computers or on additional computers that have network access.

1. Installing AudioRack on the computers
First, you will need to install the AudioRack package on the computers you plan to run ARServer, ARStudio and ARManager on.

2. Configure a file server
Second, configure the 192.168.0.1 machine as a network file server. You can use Apple's File sharing, NFS, Window/SMB or any Mac compatible file sharing protocol. However, to provide specific instructions in this guide, we will be assuming you are using AFP (Apple's file sharing) in the examples that follow. Set up a shared disk or disks on this system and copy all your audio files onto that share point or points. Details on how this is done is beyond the scope of

this guide and will vary depending on what sharing protocol/software you use. This machine need not be running MacOS as long as it can server files to MacOS computers. Test your shared disks by mounting them on one of the other MacOS computers. You can use the Finder (Go menu -> Connect To Server) to connect to the server at 192.168.0.1 if you are using Apple's File Sharing or SMB. Make sure you can access the disks and the audio files. Watch out for file access privileges when you configure the file server. You will want to give EVERYONE read access to the disks and audio files unless you want to contend with either setting up matching unix user IDs and account across the machines or using Kerberos Services to achieve the same goal. You can set up a fire wall to prevent outside networks from accessing the files if you are concerned about security with public/everyone read access.

3. Configure a database server
Third, you will need to have MySQL installed, configured and running on the 192.168.0.2 machine. This computer doesn't need to be running MacOS, as long as it can run MySQL server. If you followed the instruction in section 2.11, you could make the computer on which you did the initial installation the MySQL server, as it is already up and running with a properly configured library. Otherwise, you will want to create a new Library as was done in section 2.1.1 AFTER the next step, where we get ARManager running. In addition, there are various tool available on the internet to perform database dumps into a file and then reload the dump onto a different server if you want to move the library from the section 2.1.1 example on to a new computer. You can also do this from the command line based MySQL client included with the MySQL installation. Have a look at the MySQL documentation for details.

4. Modify library for new audio file locations
Fourth, run ARManager on the "192.168.0.4" computer. At the connection panel, enter 192.168.0.2 for the Server Address instead of "localhost" as was done before. This tells ARManager to connect to the 192.168.0.2 machine for database access. Enter the password, username, etc, as required by the database server. Now you can follow the steps to create a new library from section 2.1.1, or if the library already exists on the server, you can just click the **Apply** button. You can save the settings as a Favorite using the same procedure outline in section 2.1.1 so you don't have to reenter the settings next time you run ARManager. You now need to mount the shared disks on the 192.168.0.1 machine on this computer. This will allow you to add audio files from those disks to the library, making sure that other computers can access the files as well. You can now add new audio files to the library as was done in section 2.1.1, making sure to **only add files from the shared disks.** If the library already contains files that used to be on another disk and you moved them onto the shared disks, you can re-add all the files (just drag the shared disks into the list in the **Import Audio Files** panel to add all the audio files on the disks). Before you click the **Import** button, check the **File Hash Signature Match** button and the **Update File Location** button. This will cause the import function to update the file path, mount, and File Hash properties in the library of all the audio files you are re-importing rather than having duplicate entries, of which the original files will not be findable. That's it for ARManager configuration. Use ARManager to browse, edit, view, and manage the library and schedule as before. Install it on as many computers as you like with the same configuration, again using the shared disks to access the files themselves if need be.

5. Configure ARServer
Fifth, you will want to get ARServer running on the 192.168.0.3 computer. We will assume you want it to run at system startup time. This complicates things a bit because ARServer will need access to the shared disks on the file server as soon as it is running. You will need to make this computer mount the shared disks as a static mount which is available to all users on the computer just after the system boots up rather than being available only after a user connects to the server through the finder. The required steps seem to change as Apple releases new operating systems. As of OS X 10.6, the procedure is as follows:

From the terminal application window type

sudo pico/etc/auto_master

Add the following line to the file to use afp (Apple File Sharing) protocol:

**/-		auto_afp**

Or add the following line to the file to use smb (Windows/Samba File Sharing) protocol:

**/-		auto_smb**

Save the changes.

This will cause a file named auto_afp to be read from the /etc directory when auto-mounted disk are loaded into the system file hierarchy. The mounts described in the referenced file will be placed in the root directory, so in the file we will need to specify the full path to the mount.

Now, we create either the auto_afp file or auto_smb file :

sudo pico/etc/auto_afp

Add the a line formatted as follows to the file for each shared disk you want to mount, note the spaces:

/private/var/automount/Network/MountName -fstype=afp afp://username:password@server-address/ShareName

If for example, a file server's address is 192.168.0.1, with shared disk named is AudioDisk, and you want the share to be called ServerAudioDisk on your system, then the new line would be:

/private/var/automount/Network/AudioDisk -fstype=afp afp://username:password@192.168.0.1/AudioDisk

Note the full path specified at the beginning of the line. This path is the first location in the ARServer default volume search list. If you choose to have the mounts places in some other location, be sure to include that location in the "file mount prefix list" in the library tab of ARManager.

For the smb protocol:

sudo pico/etc/auto_smb

Add the a line formatted as follows to the file for each shared disk you want to mount:

/private/var/automount/Network/MountName -fstype=smbfs ://username:password@server-address/AudioDisk

Again, if the file server's address is 192.168.0.1, with shared disk named is AudioDisk, and you want the share to be called ServerAudioDisk on your system, then the new line would be:

/private/var/automount/Network/AudioDisk -fstype=smbfs ://username:password@192.168.0.1/AudioDisk

In both cases, the username and password would be the username and password **on the file server** who's access privileges you would like to use to access the shared disk. Note that once the disk has been mounted into the local computers file system, the first user on the local computer to access the disk will become the local owner of the disk. If you mount the disk, then access it from a user account prior to ARServer using it,the account you accessed it from will become the owner; ARServer my not be able to access it, depending on the public permissions set for the disk and folders at the server side.

Save the changes, and issue the following terminal command to update the mounts with out having to reboot the computer:

```
sudo automount -cv
```

You can now access the automounted disk at the following location: /private/var/automount/Network/ServerAudioDisk.

Finally, we will configure ARServer, but we will do that from ARStudio, which will be running on the fifth computer. Run ARStudio as in the previous section, only this time, open the preferences panel and click on the **Server** tab. Click the **Remote** button and enter "192.168.0.4" for the Server Address and "9550" for the Server Port (this is the default port number). Click the Apply button, and the status at the top should report: "Connected to 192.168.0.4:9550". You are now connected to the remote ARServer instance running on the 192.168.0.4 machine. Click the Configure Server button to open the ARServer configuration panel (now for the remote ARServer instance) and set up ARServer as is outlines in section 2.1.1 with one exception: The database server should now be set to "192.168.0.2" instead of localhost. You can also copy and modify existing configuration files instead of reconfiguring the ARServer instance from scratch as outlined in section 2.1.1 optional step 5.

When you are done, save the settings and close the ARServer configuration panel. You may want to save your remote ARServer connection settings to a favorite before you close the ARStudio preferences panel. Enter a name for the favorite in the field next to the **Add:** button. The button will become enabled when a name is in the field. Click the button to save the settings under that name in the **Favorites** list to the left so you do not need to reenter the address and port number next time you run ARStudio.

The multi-computer configuration is now complete. You can now use ARStudio and ARManager on this or any of the other computers to manage ARServer,

2.2 Multi-Station

In Multi station mode, multiple ARServer instance will share a common audio library, all with different automation schedules and logs. In addition, the multiple ARServer instances (one for each station) can be running on the same host computer (shared computer) or they can be running separately on different machines (separate computers). As was possible with the single station mode (section 2.1), you can run the user interface components (ARStudio and ARManager), the database server and the file system on computers other that the machines that will be running ARServer. Review section 2.1.2 for details.

Each station will share a common audio library but will use different "locations" with in that library to separately store their schedules and program logs. First, we will configure the audio Library, as this step is the same regardless of how many computers you plan to use to run the stations. We will assume you have a library already set up and running on a database server from section 2.1 The database server can be on the same machine you intend to run the stations on or on a different machine altogether as was done in section 2.1.2. Run ARManager to access the library database. Assuming you have a single location with automation already set up in that library, simply select that location in the **Locations** list of the **Manage** tab of Library window. Under the **Edit** menu, select **Duplicate**. The location, with all it's automation properties has just been duplicated into a new location with the same name as the original with "copy" added to the end. Rename the location if you like by double clicking the copy location in the list and entering a different name. Repeat for as many locations as you need, one for each station.

You can now select each of the locations one at a time and edit the schedule, create new pick tasks, new playlist rotations, etc, as required to set each of the station's automation as you desire.

2.2.1 Separate computer per station

In this configuration, there will be only one instance of ARServer running on a computer at a time, as was true in the section 2.1. Multiple station are supported by having a computer for each station with each of those computers sharing one common database library. Since the audio hardware used by ARServer is tied to the computer on which ARServer is running, this also implies that each station will have it's own audio hardware.

You will need to do the following:

1. Install The AudioRack Suite software on each of the target computers that will be running ARServer.

2. Configure ARServer on each of the target computers.

Follow the directions outlined in section 2.1.2 in step five to configure ARServer on each machine. Use the IP address of the target computer in place of 192.168.0.3 address used in that example. In addition, when configuring the ARServer database settings from the ARStudio application, set each of the ARServer instances to a library different location (as created in section 2.2) corresponding to the station you want that ARServer instance to run.

You now have each ARServer computer running a separate, independent station. You can connect to any of them from ARStudio running on any of the networked machines by entering the appropriate server address, and you can manage the schedule for each with ARManager on any of the networked computers by selection the desired location from **Manage** tab of the library window, and proceeding to view/edit items and there schedule.

2.2.2 Shared computer for multiple stations

This configuration is the same as that in 2.2.1 except all the ARServer instances for each station will be running at the same time on a single computer. The first thing to consider is audio hardware. Since there is only one computer and multiple stations, you will probably need audio hardware connected to the computer that supports multiple stereo inputs and outputs so each station can have it's own pair of output channels on it's main mixer bus. If you are only using ARServer to encode a shoutcast or rsp stream, you can use the built-in audio hardware for all the stations assigning the built-in audio output to each of the ARServer instances cue or alternate mix bus or some other mix bus you don't plan to use. Recall that ARServer requires at least one output group defined which is tied to audio hardware for it to use as a master sample rate clock - even if the output is just carrying silence from an unused output bus.

After you have decided what you require for audio hardware, install AudioRack Suite on the target computer. We will assume the target computer has an IP address of 192.168.0.3 (same as used in the 2.1.2 example). Go to the **Applications** folder on the system disk of the target computer, open the **ARSuite** folder and run the applescript called **ServerStartupEnable.app**. You will be asked for an administrators password as the script runs. When it is finished, ARServer will be running under the arserver user account, and it will start automatically when ever the system is rebooted. Configure this instance of ARServer using ARStudio as was done in section 2.2.1. You now have one instance configured and running.

The following steps will be repeated for each instance of ARServer you would like to run in addition the the current instance.

1. Create a directory for and edit each instances' configuration files
This will require that you run the Terminal Application from an Administrator account on the target computer.

Once the application is running, type the following at the prompt:

sudo cp -Rp /etc/arserver /etc/arserver9551

You will be asked for the administrators password. Enter it to proceed. Note also that 9551 added to the end is the control port number this instance will be set to use as to not interfere with the first instance which is running on port 9550 (the default port number). You will want to increment this number by 1 for each additional instance you create, and use that number in all places that follow where a port number is specified.

next type the following:

sudo pico /etc/arserver9551/ars_startup.conf

This will run the pico text editor in the Terminal Application window. Find the following line in the file:

-p 9550

and change it to

-p 9551

This will make the ARServer instance using this configuration run on control port 9551.

Next, find all the places which "/etc/arserver/" appears and replace it with "/etc/arserver9551". For example, the following line will change from:

config /etc/arserver/ars_output.conf

to:

config /etc/arserver9551/ars_output.conf

When your done changing all the lines with "/etc/arserver" in them, save the file by pressing the control-o key and hitting the return key. Hit control-x to quit the pico editor.

2. Create and configure a startup item for the instance.
Again, from the Terminal Application, with in an Administrator account, type the following lines:

sudo cp -p /Library/LaunchDaemons/com.redmountainradio.arserver.plist /Library/LaunchDaemons/com.redmountainradio.arserver9551.plist

and then run the pico editor again:

sudo pico /Library/LaunchDaemons/com.redmountainradio.arserver9551.plist

change the line that reads:

```
<string>com.redmountainradio.arserver</string>
```

to:

```
<string>com.redmountainradio.arserver9551</string>
```

change the line that reads:

```
<string>/etc/arserver/ars_startup.conf</string>
```

to:

```
<string>/etc/arserver9551/ars_startup.conf</string>
```

When your done, save the file by pressing the control-o key and hitting the return key. Hit control-x to quit the pico editor.

Finally, type the following to start the new instance of ARServer running along side the other instance:

sudo launchctl load -w /Library/LaunchDaemons/com.redmountainradio.arserver9551.plist

The new instance will start running, and will also do so automatically next time you restart the computer. To disable it from starting, rename the “arserver9551” file to “.arserver9551” using the following terminal command:

sudo launchctl unload -w /Library/LaunchDaemons/com.redmountainradio.arserver9551.plist

You can connect to it using ARStudio as before but specify the new port number 9551 when you connect to connect to this instance instead of the other instance running on port 9550. The new instance will be configured exactly the same as the original, so you will only need to change database locations and audio hardware settings as everything else is already configured from the original.

Repeat the steps above for each additional ARServer instance/station you want running on this single computer using control port numbers like 9552, 9553, etc. You now have several stations running from a single computer. Remember the port number assigned for each station set up so you can connect to the desired station with ARStudio.

3.0 Operation

3.1 ARStudio

coming soon.

3.2 ARManager

coming soon.

Appendix A ARServer commands

System/Configuartion Commands

`info`
shows version information, etc.

`stat`
returns the current general status of the playlist, automation, logs, etc.

`settings [key string (optional)] [value string(optional)]`
Returns all the current setting (automation, database, etc.) keys and values.

`get [key string]`
Returns the setting value (automation, database, etc.) for the specified key, the value for that key will be

`set [key string] [value string]`
Sets the specified setting key (automation, database, etc.) to the specified value.

`setpath [path string]`
sets the save path to the settings configuration file.

`saveset`
saves all the current settings to the settings configuration file, if the `setpath` path has been set.

`config [path string]`
loads/executes a configuration file. Partial paths are interpreted relative to ARServer's working directory when it was launched.

`clients`
shows a list of the currently connected client ip addresses.

`echo [message string]`
prints a message to back out the sending control port.

`dblist`
prints a list of the (libdbi) database drivers installed on this machine.

dbsync
Scans through the files in the current database, updating Hash and mount information for each file URL. This function is useful for updating the database in some cases when directed by the version upgrade directions.

dbfilesearch [optional path string]
Scans through all the files in either the specified file path tree (if given) or all the mount found in the database, checking all the file Hash code and mount information against the database to locate any files that may have been moved or renamed. The system setting db_file_search_pace sets the delay, in milliseconds, between each file checked; or 250 mS be default.

dbinit
sets up a the database structure of a new database for use. Settings for db_name, db_type and (server) db_user, db_pw, db_server or (SQLite) db_dir must point to the new database.

inpath [path string]
sets the save path to the line input definition configuration file.

outpath [path string]
sets the save path to the audio output setup configuration file.

tasks
returns an indexed list of all the running tasks.

deltask
deletes/stops the specified (by index) task from running.

execute [command string]
executes (via fork and exec) the specified unix shell command.

attach [command string]
same as execute, except the forked process stdin and stdout streams are attached to the issuing session streams and control is not returned to ARServer until the process is complete.

modbusset IP-address UnitID CoilAddress
Send a modbus/TCP coil on command to the modbus interface at the specified IP-address (must be numeric, not named) on port 502. UnitID is one byte hex (usually 01), CoilAddress is two byte hex.

modbusclear IP-address UnitID CoilAddress
Send a modbus/TCP coil off command to the modbus interface at the specified IP-address (must be numeric, not named) on port 502. UnitID is one byte hex (usually 01), CoilAddress is two byte hex.

modbuspoll period IP-address UnitID InputAddress config_file
Creates a non-expiring task that runs in the background (see task functions) to pool the specified modbus device's digital input. When the polling discovers that the input has changes state, the specified configuration file (found in the directory specified in the system setting file_trigger_dir) with either .on or .off appended to the file name to representing the input's new state is executed. Period is an decimal integer indicating the polling period in seconds. IP-address of the device to poll must

be in numeric, not named, on port 502. UnitID is one byte hex (usually 01), InputAddress is two byte hex.

iaxstream
Starts a tcp<->udp tunnel on the controlling terminal with udp destination set to the local computer on the iax port. This command is intended to allow iax telephone usage by a remote ARStudio client. Control WILL NOT return to ARServer on the connection used to issue this command. If you plan to use this functio, you should open a new connection to ARServer just for this purpose.

vuon
vuoff
turns on or off notification vu meters reporting for a client after notification has been enabled.

notify
registers the connection this command was issued on to receive server state change notice packets.
notification packets are of the form (multi-byte values are sent MSB first):

```
struct notifyConatiner {
      char                  marker;    // 0 - flags the start of a notify packet
      char                  type;      // see notice type codes below
      unsigned short        dataSize;  // byte count of the data that follows
      char                  data[1];   // var length, notifyData or vuContainer
};

struct notifyData{
      unsigned long         senderID;
      long                  reference;
      union {
            float           fVal;
            long            iVal;
            unsigned char   cVal[4];
      } value;                         // the data value
};

struct notifyFXParam{
       unsigned long AID;
       unsigned char count;
       float  value[1];                // array of MeterReadOnly parameter values in ascending
                                       // order by parameter ID: First through Nth entry is parameter value of
                                       // lowest through highest parameter IDs with the parameterInfo flag
                                       // kAudioUnitParameterFlag_MeterReadOnly set.
};

struct vuContainer {
      char            count;           // number of vuInstances that follow
      char            data[1];         // var length, array of vuInstances
};

struct vuInstance {
      unsigned long   uid;             // 0 for output buses or player/recorder UID
      char            count;           // number of channels in vuData array
      char            data[1];         // var length, array of vuData structures
};

struct vuData {
      unsigned char   peak;      // value = 255 times sqrt of scalar magnitude
      unsigned char   avr;             // value = 255 times sqrt of scalar magnitude
};
```

```
notice type codes (hex):
0x01 - volume (data: notifyData float)
0x02 - balance (data: notifyData float)
0x03 - bus assignment (data: notifyData integer)
0x04 - player status bits (data: notifyData integer)
0x05 - general status change (data notifyData 0) - client should get new status.
0x06 - metadata change (data: notifyData 0) - client should get new metadata for this UID.
0x07 - recorder status change (data: notifyData 0) - client should update all recorder status.
0x08 - output and input vu meters data - (data: vuContainer)
0x09 - recorder gain changed (data: notifyData float)
0x0a - player position changed (data: notifyData float)
0x0b - item deleted (data: notifyData 0)
0x0c - delay settings changed - (data: notifyData 0) client should get delay status
0x0d   - AudioUnit realtime parameter - (data: notifyFXParam)
0x0e - AudioUnit change - (data: notifyData integer = bus, reference = AID) - client should use fxslots command
0x0f - AudioUnit cue change - (data: notifyData integer = bus, reference = UID or zero for effects cue diabale)
0x10 - Processor load values - (data: notifyData char[0] = total system load,  char[1] = arserver load)
```

iaxinit
shutdown and re-initializes the iax telephone system using the current iax_ group of key/value settings (see 'settings' command).

exit
close this control session (remote connection). Notification to this client (if enabled) is stopped also.

restart
Shuts down ARServer and closes all control sessions. If ARServer was run in keep-alive mode, it will be restarted

shutdown
Shuts down ARServer and closes all control sessions. No restarting even if run in keep alive mode.

Audio Device Control Commands

meters
returns the current peak and average VU meter readings for each output channel

devlist
returns information on all the audio input and output devices in the system

setin [input-name string] [bus hex] [left integer] [right integer] [NMC-ID integer] [available-controls hex] [device string (UID)]
creates or updates a line input definition mapping the left and right channels to the given device input numbers and enabling the selected output buses. Bus number bits 0 through 23 are stereo mixer output bus enables, bits 24 through 26 are mute group A, B and C enables. Mute group is enables/disabled at play/stop. NMC-ID is the Network Machine Control device ID (0-255) for sending network machine control messages (i.e. start, stop, etc.) to a controller associated with this input.

available-controls is used to notifies clients what controls the input supports setting a bit indicates that the corresponding control is supported:

0-volume, 1-position, 2-skip ahead, 3-skip back, 4-fade, 5-feed.

delin [name string]
deletes specified line input definition.

getin [name string]
returns specified line input definition.

dumpin
returns a list of all the line input settings.

inuid [name string]
returns the UID of the named input if it is currently loaded in a player or is in the playlist queue. Return 0 if the named input is not found.

savein
saves the current line input definitions to the line input definition configuration file if the path to the file has been set and the file permissions allow writing.

setout [name string] [type string] [muteGain integer] [bus-count integer] [bus-mapping output 0 integer]...[bus-mapping output n integer] [deviceUID string]
creates or updates the named output device definition mapping specified mixer output bus to the given device output numbers and setting the device type to either 'master' or 'slave'. There can only be one master device. By setting a device to master, the current master device, if any will become a slave. The master device has the lowest latency and provides the sample rate clock to the ARServer mixer. The muteGain integer (hex format) sets the output device gains when cue (LSB), mute-A, mute-B, or mute-C (MSB) mute groups are enables. 00=off, ff=mute attenuation gain = $((\text{hex val}) / 255)^3$.

delout [name string]
deletes specified output device definition.

getout [name string]
returns specified output device definition.

outvol [name string] [volume float]
sets the specified output device volume (1.0 = unity gain, scalar).

setdly [destination (name string or recorder UID)] [delay float (in seconds)]
sets the specified output destinations delay in seconds (0 to 10).

dump
sets the delay to zero of all output destinations.

getdly
returns a list of all the output destinations (output devices and recorders) and their delay in seconds (0 to 10).

dumpout
returns a list of all the output device definitions.

saveout
saves the current output device definitions to the output device definition configuration file if the path to the file has been set and the file permissions allow writing.

`setmm [name string] [count integer] [input-name 0 string]...[input-name n string]`
sets the list of inputs which will be mix minus-ed from the for the specified output device's monitor bus.

`getmm [name string]`
returns a list of inputs which will be mix minus-ed from the for the specified output device's monitor bus.

`setdm [source channel integer] [destination ID integer] [device string (UID)]`
Sets the direct monitoring destination for the specified source channel of an audio input device to the given destination ID, or 0 to disable. The source is the device's input channel number and the destination is an available dm destination obtained via the 'listdm' command.

`getdm [device string (UID)]`
returns a per channel list of the direct monitoring destination ID and Name (if any) set for the given audio device.

`listdm [source channel integer] [device string (UID)]`
returns a list of possible direct monitoring destinations (if any) for the specified source channel of a given audio input device. A text description (if available) and a destination ID (integer) for use with other dm commands, will be returned.

Note: If an output device supports dm, ARServer can control the dm volume and panning to mirror ARServer channel mixer settings.

`innames`
returns a list (if any) of a device's input channel names.

`outnames`
returns a list (if any) of a device's output channel names.

Player Commands

`pstat`
returns the current status and other information for each player. Status integer: bit flags b0-loading, b1-standby, b2-playing, b3-hasPlayed, b4-finished reading file, b5-logged, b6-waiting, b7-position has changes

`setstat [pNum integer] [status integer]`
sets the specified player number status integer: bit flags b0-loading, b1-standby, b2-playing, b3-hasPlayed, b4-finished reading file, b5-logged, b6-waiting, b7-position has changes

`load [pNum integer] [url string]`
loads the specified player number with the specified resource. URL is a reference to the resource to load, for example:
file://localhost/url_encoded_unix_path_to_audio_file
input:///input_name (see setin and dumpin)
item://database_item_id# (for database items)
iax:///line_number (for iax phone lines)

`cue [url string]`
The Same as loads except the next available player is loaded and the player is placed in cue. This function returns the new player's UID in hex format AND the player number that was loaded.

`unload [pNum integer]`
unloads the specified player

`stop [pNum integer]`
stops the specified player from playing

`play [pNum integer]`
starts playing the specified player

`pos [pNum integer] [position float]`
sets the play position (seconds) of the specified player

`vol [pNum integer] [volume float]`
sets the specified player's volume (1.0 = unity gain, scalar)

`bal [pNum integer] [balance float]`
sets the specified player's balance (-1.0 = L, 1.0 = R)

`bus [pNum integer] [bus hex]`
sets the specified player's output bus (set bit enables the corresponding output)

`feed [pNum integer] [bus integer] [(optional)volume float]`
sets the specified player's 'feed bus' (audio sent back to the source) and feed volume (1.0 = unity gain, scalar) to the specified mixer output bus number: 0=mon, 1=cue, 2=main, 3=alt (iax phone line players only).

`fade [pNum integer] [time float]`
sets the time position to start fading out the specified player number

`next [pNum integer] [next integer] [time float]`
sets the time position for the specified player (pNum) to start playing the next player (next)

Playlist Commands

`list`
Dumps the current play list queue

`add [pos or UID integer] [url string]`
`add [pos or UID integer] [pNum integer]`
inserts the specified item or player into the play list at index (decimal) pos (-1 for end of list, -2 for next in list) or at the same position as the list item with (hex) pos UID.
Note: url supports stop:/// for a playlist stop.

`uadd [pos or UID integer] [url string]`

Unique add: The same as add above except it does not accept player numbers and will only add the item if the url is not yet in the playlist. If an item with the specified url is already in the playlist, it simply sets lastuid to that items uid value and returns the value.

`split [UID integer] [url string]`
inserts the specified (by URL) item into the play list at the same position as an existing list item with the specified (hex) UID or -1 for the last used UID or the parent UID. The new item inherits the exiting item's meta properties for def_segout, def_seglevel, SegIn, Volume, Duration, FadeOut, Priority, TargetTime, FillTime and FadeTime.

`segnow`
If an item is currently playing and is set to segue into another item, this command will forces it to immediately fade out and segue into the next item.

`segall`
This command will forces all playing items from the playlist to immediately fade out and segue into the next non-playing item in the list.

`delete [pos integer]`
deletes the play list item at index (decimal) pos ot UID (hex) pos. If the item is in a player, it will be unloaded too.

`move [from-pos integer] [to-pos integer]`
moves the play list item at index from-pos to index to-pos if decimal or UID to UID in hex format.

`run`
starts the automatic loading and playing of items in the playlist queue.

`halt`
stops the automatic loading and playing of items in the playlist queue. Items currently loaded into a
	player unloaded unless thay are playing.

`expand [UID integer]`
Starts the expansion of a file or database playlist item in the playlist into it's containing items.

Item metadata Commands

`urlmeta [url string]`
returns a list of metadata (if any) for the item referenced from the given url.

`setmeta [metaUID or pNum integer] [key string] [value string]`
set specified player number (decimal) or UID (hex) meta data key/value pair

`delmeta [metaUID integer] [key string]`
deletes specified player number (decimal) or UID (hex) meta data key/value pair

`getmeta [metaUID integer] [key string]` returns specified player number (decimal) or UID (hex) meta string for the given property key.

`dumpmeta [metaUID integer]`
returns a list of all the meta data key/value pairs for the specified UID

`logmeta [metaUID integer]`
Updates Name, Artist, Album, Comment, Owner, Source properties in the program log to match the specified UID current properties where the program log record ID = UID's property logID.

`logsync`
Program logs have been added/changed by the client directly through the database... This command will send a notice to all clients to sync to the new log entry.

`getuid [property string] [value string]`
returns the UID of the first item found that has a specified metadata property value set to the specified value. This function first searches the players in ascending order, then searches the playlist in next to last order. Return 0 if none not found.

Recorder/Encoder Commands

`rstat`
returns the current status and other information for all the current recorders
`newrec`
Creates a new file recorder/streamer instance with no specified properties. The UID of the new instance is returned to reference it in the commands that follow

`initrec [UID integer]`
Initializes the specified file recorder/streamer instance UID, in hex format or -1 for last used UID. MetaData properties for the specified UID, set prior to issuing this command, are used to set up and initialize recorder/encoder. See appendix C for details on the recorder properties.

`startrec [UID integer]`
starts or re-starts the recorder with the associated UID (hex format)

`stoprec [UID integer]`
stops the recorder with the associated UID (hex format)

`recgain [UID integer] [float gain]`
Sets the recorder with the associated UID (hex format) input gain scalar.

`lockrec [UID integer]`
Locks the recorder with the associated UID (hex format) so no changes can be made to it.

`unlockrec [UID integer]`
Unlocks the recorder with the associated UID (hex format).

`closerec [UID integer]`

closes out the recorder with the associated UID (hex format), deleting the recorder instance.

Audio Processing Commands

`fxlist`

shows all the AudioUnit (AU) type audio effects processor plugins available on the system that ARServer can use. ARServer requires all AUs to support 2 channels in and out.

`fxslots [UID/OID integer] [bus integer]`

returns the status of all the audio processing slots associated with a device. UID/OID is the device specifier. The device specifier for players and recorders is the player/recorder metadata UID in hex format. For output devices, it is the output OID as shown in the results of the “dumpout” command. -1 for a UID implies that ARServer should reuse the UID last used by a command that accepts or creates UIDs (lastuid session variable). The bus parameter specifies which processing stack is to be examined. File and input players have only one processing stack attached to their stereo outputs, bus 0. IAX telephone players have two busses, bus 0 attached to their stereo outputs and bus 1 attached to the audio feed sent back to the caller. Output devices have the same number of busses as the mixer core, 4 by default: Bus 0 attached to the Monitor bus, bus 1 attached to the Cue bus, bus 2 attached to the Main bus and bus 3 attached to the Alt bus. Each stereo bus has it’s own processing stack associated with an output device.

`fxinsert [UID/OID integer] [slot integer] [bus integer] [subtype string] [mfg string]`

Sets the AudioUnit (AU) effects processor plugin inserted into the audio chain of the specified device, bus and slot. UID/OID is the device specifier. The device specifier for players and recorders is the player/recorder metadata UID in hex format. For output devices, it is the output OID as shown in the results of the “dumpout” command. -1 for a UID implies that ARServer should reuse the UID last used by a command that accepts or creates UIDs (lastuid session variable). There are eight slots available to each device’s bus where audio is processed in sequence of slot. For example, a compressor in slot 0, and a filter in slot 3 will result in the audio first being compressed, then filtered. File and input players have only one processing stack attached to their stereo outputs, bus 0. IAX telephone players have two busses, bus 0 attached to their stereo outputs and bus 1 attached to the audio feed sent back to the caller. Output devices have the same number of busses as the mixer core, 4 by default: Bus 0 attached to the Monitor bus, bus 1 attached to the Cue bus, bus 2 attached to the Main bus and bus 3 attached to the Alt bus. Each stereo bus has it’s own processing stack associated with an output device. The subtype and mfg parameters are used to specify the AU to insert. The “fxlist” command provides a list of available AUs and their corresponding subtype and mfg codes. If subtype and mfg are left blank, the processing slot is made to be empty with no processing. The session lastaid variable will be set to the AID of the AU slot/bus/device after this command has executed.

`fxbypass [AID integer] [value integer]`

Sets the bypass state of the specified AudioUnit (AU) effect processor. If Value is 1 the AU will be bypassed, if it’s 0 the AU will be online. AID is the AudioUnit Identifier of the AU who you want to change the bypass stat of. AID is formed from the UID/OID of the host device along with a device bus and effects processing slot specifier in the following hex format:

USBUUUUU where U is UID hex digit, S is slot digit, B is bus digit

```
examples:    Player/Recorder UID 0001a33c, slot 3, bus 1 --> AID = 0311a33c
             Output OID c000000b, slot 7, bus 3 --> AID = c730000b
```

-1 for an AID implies that ARServer should reuse the AID last used by a command that accepts or creates AIDs (lastaid session variable).

`fxparam [AID integer] [paramID integer (optional) [value float (optional)]]`

Gets or Sets the values of the specified AudioUnit (AU) effect processor's parameters. If ParamID and Value is excluded, The command will return a list of all the AU's parameters, by parameter ID, including name, value, allowed range, unit types, etc. If ParamID is specified, then the value of just that parameter is returned. If a value is specified as well, then the value of that parameter is set by the command.

AID is the AudioUnit Identifier of the AU who's parameters you want to view or manipulate. AID is formed from the UID/OID of the host device along with a device bus and effects processing slot specifier in the following hex format:

USBUUUUU where U is UID hex digit, S is slot digit, B is bus digit

```
examples:    Player/Recorder UID 0001a33c, slot 3, bus 1 --> AID = 0311a33c
             Output OID c000000b, slot 7, bus 3 --> AID = c730000b
```

`fxvalstr [AID integer] [paramID integer]`

returns the strings associated with the index values (0 to n) for AudioUnit parameters that are lists. AID is explained previously.

`fxsave [UID/OID integer] [name string]`

Saves the current audio effects settings (all slots, buses and parameters) of the specified device to a configuration file of the given name with in the effect configuration directory. The effect configuration directory is set by the ARServer setting "file_fx_dir" and is by default the same directory that contains the startup.conf file. The resulting configuration file can be loaded with the "config" command to restore the settings of a device as saved if the lastuid session variable is first set to the desired device to restore.

There are several default ARServer configuration file names used by ARServer:

ars_player_default.fx	default effects settings for file players
ars_recorder_default.fx	default effects settings for recorders
ars_iax_default.fx	default effects settings for iax telephone players
ars_iax_aac.fx	default effects settings for iax aac players (ARStudio feed)
ars_input_[name].fx	Live input player where [name] is the input name
ars_output_[name].fx	Output group where [name] is the output name

If an effects configuration file is created with the fxsave command having any of these names, ARServer will load the corresponding file, applying it a new device when a device of that type is loaded/initialized. If no configuration file is found for a device, then the device is loaded/initialized with all it's effects slots empty.

The above device default effect configuration files can be overridden by setting a player/recorder's "fx_config" metadata property to the desired alternate configuration file name prior to the player/recorder loading/initializing.

fxpreset [AID integer] [value integer (optional)]
Returns a list of factory presets available for the specified AU by name and preset index, or if value is included, set the specified AU parameters to that of the factory preset index specified as the value. AID is explained previously.

fxcue [UID integer [bus integer] (optional)]
Places the specified device's effects chain into cue mode, allowing the output audio (post audio effects processing) from the slots to be monitored through the mixer cue bus. If no device UID and bus is specified, then effects cueing is disabled. Only a single effects chain can be in cue at a time. Specifying a new device and bus will transfer cue mode to the new device and bus. Cue mode does not interfere with the effects chain's normal operation - audio will continue to feed the chains destination.

fxwatch [AID integer (optional)]
The session from which this command is sent will have notifications enable for effects parameter changes related to realtime parameters (level indicators for example) of the AU corresponding to the specified AID. If no AID is specified, effects parameter changes notifications are disabled for this session.
NOTE: This command can result in effects parameter change notifications being sent to a session even if other notifications are otherwise disabled.

lastuid [UID integer]
Sets the session lastuid variable to the specified UID such that subsequent commands that pass -1 for an UID parameter will use the specified UID. This is useful if you want to load a saved effects configuration file for a device using the config command. Saved effects configuration files assume lastuid is set to the UID of the device (player/recorder/output) before the config command is issues.

lastaid [AID integer]
Sets the session lastaid variable to the specified AID such that subsequent commands that pass -1 for an AID parameter will use the specified AID. AID is explained previously.

Appendix B ARServer settings

ARServer stores most of it's settings as text bases key-value pairs. The following is a list of setting that ARServer pays attention to. All these settings are stored in a special metadata record having a UID of 0. The settings are ALL accessible using the ARServer control console commands **settings**, **get** and **set** (see appendix A). Many, but not all, of these properties are also accessed via the ARStudio user interface, via the server settings panel in the preferences window. In the latter case, the settings are viewed and changes via graphical controls such as text fields, sliders, check boxes, etc. If you access any of the settings via the ARServer control console, you can issue the **saveset** command to re-save the settings into a configuration file which is loaded next time ARServer is started.

Property key	Value type	Function	User Interface
auto_live_flags	decimal int	Enables live mode automation behavior based on the bits being set in the binary number equivalent: b0 Playlist Filling enabled b1 Schedule Inserting enabled b2 Playlist Queue stopping allowed b3 Target time re-ordering enabled	Yes
auto_live_timeout	decimal int	Minutes of no user activity needed for automation to return from live to auto	Yes
auto_startup	bool (0 or 1)	If set to 1, automation is auto mode when ARServer is first run, other wise it's off	Yes
auto_thresh	decimal int	Minimum number of items the ARServer list filler try to keep in the playlist queue when automation is on	Yes
client_players_visible	decimal int	ARSuite will not show player numbers greater than this number in it's player area.	No
db_dir	string	SQLite library database director path	Yes
db_include_loc	string	Coma separated list of Library location ID who's log play history are to be considered along with the current location (see below) for automation picks	No
db_loc	string	Library location ID for logging and automation history	Yes

Property key	Value type	Function	User Interface
db_mark_missing	bool (0 or 1)	If set to 1, ARServer will flag missing files in the library that it comes across (excluding them from automation picks) and keep the location of moved files up-to-date in the library.	Yes
db_name	string	Library database name	Yes
db_prefix	string	Library table name prefix. "ar_" by default	Yes
db_pw	string	Library database access password	Yes
db_server	string	Library database server address	Yes
db_type	string	Library database type	Yes
db_user	string	Library database access user name	Yes
def_bus	decimal int	Sets file player output bus assignments based on the bits being set in the binary number equivalent: b0 Monitor Bus b1 Cue Bus b2 Main Bus b3 Alternate Bus	Yes
def_record_dir	string	File path to default director where recordings are saved	Yes
def_segout	decimal float	Items with no segout time specified will use this time, in seconds, from the end of the item as the segout time.	Yes
def_seglevel	decimal float	Default scalar level (RMS and channels averaged) to trigger a segue AFTER the segue time. Set to zero to ignore level.	Yes
file_encoders	string	Location of the directory which contains the LAME/mp3 encoder	No
file_fx_dir	string	Specifies the path to the directory that contains configuration files for player/recorder/output effects processing settings. See the appendix A **Audio Processing** section for details.	
file_inputs	string	File path of the audio input definition configuration file	No
file_log	string	File path of the server log file	No
file_outputs	string	File path of the audio output configuration file	No
file_prefs	string	File path of the ARServer preferences (settings) configuration file	No
file_trigger_dir	string	File path to directory containing MODBUS relay control scripts (see appendix G)	No

Property key	Value type	Function	User Interface
iax_auth_pw	string	Password required for incoming call to be accepted	Yes
iax_auth_user	string	Username required for incoming call to be accepted. If not set or blank, call authentication is disabled.	Yes
iax_control_number	decimal int	IAX calls received by ARServer which were placed to this number are granted control ability: ARServer will execute macro when DTMF keys are pressed by the caller	Yes
iax_def_bus	decimal int	IAX player output bus assignments (audio from caller) based on the bits set in the binary number equivalent: b0 Monitor Bus b1 Cue Bus b2 Main Bus b3 Alternate Bus	Yes
iax_def_feed	decimal int	Feed bus number (output bus) sent back to a caller for an IAX player when loaded after a call is received 0 Monitor 1 Cue 2 Main 3 Alternate	Yes
iax_def_feed_vol	decimal float	Feed volume setting (scalar) back to a caller for an IAX player when loaded after a call is received	Yes
iax_def_vol	decimal float	Player volume setting (scalar) for an IAX player when loaded after a call is received	Yes
iax_dtmf_macro_#	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_*	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_0	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_1	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_2	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_3	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_4	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes

Property key	Value type	Function	User Interface
iax_dtmf_macro_5	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_6	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_7	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_8	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_dtmf_macro_9	string	ARServer command to execute when DTMF key is pressed by a caller to the IAX control number	Yes
iax_max_lines	decimal int	Maximum number of calls the IAX system will except at a time. Set to 0 to disable the IAX system.	Yes
iax_port_number	decimal int	UDP port number for IAX sessions. Set to 0 to use default IAX port.	Yes
iax_reg_host	string	IP addres/Name of asterisk server to register with	Yes
iax_reg_pw	string	asterisk registration password	Yes
iax_reg_refresh	decimal int	Registration refresh time in seconds if last registration succeeded	Yes
iax_reg_retry	decimal int	Registration retry time in seconds if last registration attempt failed	Yes
iax_reg_user	string	asterisk registration user name	Yes
sys_autoexit	bool (0 or 1)	If set to 1, ARServer will shut down when the last client disconnects	Yes
sys_log_dropout	bool (0 or 1)	If set to 1, audio dropouts are logged in the ARServer log file	No
sys_logscript	string	Specifies the file path (if any) to an external program to be execute each time a new item is logged/played. The program is pass the ID number (decimal) of the library log entry just made and the file path of the ARServer settings configuration file, which would be parsed by the program to figure out how to access the log table in the database.	No
sys_passthru	bool (0 or 1)	If set to 1, causes the mixer to pass each player through to it's corresponding output bus number, repeating after the player number is greater than last bus number	Yes

Property key	Value type	Function	User Interface
sys_silence_bus	decimal int	Mix bus number the silence detector monitors. Default: 2 (Main Bus)	No
sys_silence_timeout	decimal int	If non-zero, consecutive seconds of silence + 60 required to trip the watchdog timer. Watchdog timer will restart ARServer if it was run with the -k flag. Default: 0 (Off)	No
sys_silence_thresh	decimal float	Level (scalar) below which the silence detector starts counting silence time. Default: 0.003 (-50 dB)	No

Note 1: Default values are the values assumed when there is NO property key set, not the value when the Property Key's value is blank. A blank value is interpreted as either 0 (for numeric values) or an empty string.

Note 2: The string "[self]" can be used in a DTMF macro to resolve back to the player number in which the DTMF key was pressed.

Appendix C Item meta data

As with ARServer settings, ARServer uses a key/value pair collection associated with each player, recorder, task, or playlist queue item to store information either it of the user may need about the item. The ARServer console command **getmeta**, **setmeta**, and **dumpmeta** are used to access this information (see Appendix A). The ARStudio application provides a graphical user interface to most of these properties as well.

All items must have the required metadata. Other items will have various combinations of metadata properties depending on what and where they are. For example, an audio input loaded into a player will have the required properties and the player properties. An audio file item loaded into a player and in the playlist queue will have the required properties, the player properties, the file player properties and the playlist queue properties.

Base meta data

Property key	Value type	Function
Added	decimal unixtime	Required only if this item came from the library. This is the date/time in unixtime format when the item was added to the library.
Fingerprint	decimal int	Required only if this item came from the library. Unique ID for the library this item came from: found in the value column of the “info” table under where the Property column = “Fingerprint”.
fx_config	string	if set prior to an item being loaded into a player/recorder, this property will override the name of the default effects configuration file used to set up the audio processing associated with the item. See the appendix A **Audio Processing** section for details. This is NOT a full path to the desired file, only the file name with in the effects configuration file directory, specified by the file_fx_dir setting. Ignored is blank or missing.
ID	decimal int	Required only if this item came from the library. Library record ID for this item found in the “toc” database table.
Name	string	Item name
Type	string	Item type: item (library), stop (playlist queue stop marker), iax (voip phone), file, input, task, playlist (library based playlist), filepl (file based playlist) recorders/encoder types: lame/mp3, aiff, wav, shoutcast

Property key	Value type	Function
URL	string	URL of the item's resource. Can be blank for tasks and recorder/encoders. examples: input:///Mic1 file://localhost/Volumes/AudioLib/Promo/Listen_at_10PM.mp3 item:///3460 stop:///

Player meta data

Property key	Value type	Function
Controls	decimal int	Enables user interface controls in ARStudio based on the bits being set in the binary number equivalent: b0 Volume Fader b1 Play position control b2 Skip forward button b3 Skip back button b4 Fade control b5 Send feed select list
Album	string	Name of the Album this item come from
AlbumID	decimal int	Library database record ID for the Album this item came from if the item is in the Library
Artist	string	Name of the Album this item come from
ArtistID	decimal int	Library database record ID for the Artist this item came from if the item is in the Library
def_bus	decimal int	Override the ARServer settings for default bus for this item. Used if the **split** command is executed against this item (see Appendix A)
def_segout	decimal float	Override the ARServer settings for default segout time. Used if the **split** command is executed against this item (see Appendix A)
Duration	decimal float	Duration of the item in seconds
FadeOut	decimal float	Play time into the item, in seconds, where a fadeout should occur. Set to 0, leave blank, or don't set to disable fadeout.
Intro	decimal float	A cue to DJ as to the playtime into the item where lyrics begin.
Memory	decimal float	[Not yet implemented] If a player is unloaded with this value set to non-zero, the next time this item is loaded into a player, it's play time will be set to this value.
Script	string	URL encoded text to be displayed for a DJ to read instead of playing the audio file.

Property key	Value type	Function
SegIn	decimal float	Time, in seconds, to early start this item playing in the playlist queue. This item will start at the SegOut time of the the currently playing item less this time. Set to 0.0, leave blank or don't set for no early start.
SegOut	decimal float	Play time into the item, in seconds, after which ARServer should start the next item in the playlist queue playing.
Track	decimal int	Track number of this item on the Album from which this item came.
Volume	decimal float	Scalar number specifying the default playback gain for this item. Set to 0.0, leave blank or don't set to use the default gain of 1 (unity gain)
logID	decimal int	Library database ID of the Log entry for this item, if any (it has been played or added to the playlist queue)

File Player meta data

Property key	Value type	Function
FileID	decimal int	File system inode number for the file
Hash	hex	Hash code for the file (see Appendix E)
Missing	bool (1 or 0)	Set to 1 if the file could not be found
Mount	string	Full path the the disk that contains the file
MountList	string	A comma separated list of file paths to possible places ARServer should look to find the disk that contains this file. This list is acquired from the Library settings. See Appendix D for details of how ARServer locates files.

Item in playlist queue meta data

Property key	Value type	Function
def_segout	decimal float	This is the time, in seconds, from the end of this item to start the next item in the playlist queue if no SegOut property is set. Taken from ARServer settings by default.
def_seglevel	decimal float	This is the scalar level (RMS and channel averaged) below which to trigger a segue after the segue time has passed. Set to zero to ignore levels for segues.
Duration	string	Library location ID
FillTime	decimal unixtime	If this item was added to the playlist queue by ARServer as a automation fill insert, this is the exact time, in unixtime format, when the item was scheduled to play.
Owner	string	Name of task or playlist that put this item in the playlist queue, if any.

Property key	Value type	Function
Priority	decimal int	If this item was added to the playlist queue by ARServer as a automation schedule or fill insert, this is the priority code it was inserted with (0 to 10).
TargetTime	decimal unixtime	If this item was added to the playlist queue by ARServer as a automation schedule insert, this is the exact time, in unixtime format, when the item was scheduled to play.
logID	decimal int	Library database ID of the Log entry for this item.

Recorder/encoder meta data

Property key	Value type	Function
Addresses	string	Comma separated IPv4 or IPv6 Address list (name, number) to which an RSP stream encoder should send stream packets.
aim	string	Shoutcast encoder: aim tag sent to server
Album	string	Album name to set the corresponding mp3 file metatag/shoutcast tag to.
Artist	string	Artist name to set the corresponding mp3 file metatag/shoutcast tag to.
Bind	string	IP address from which the RSP stream packets should come from if computer running arserver has more than one IP address. Set to 0.0.0.0 for IPv4 and :: for IPv6 to use any available address.
BitRate	decimal int	Shoutcast , RSP and compressed file bitrate in kilobits per second
Bus	decimal int	If the recording/encoding source is the mixer, this property specifies which mixer output bus to use: 0 Monitor 1 Cue 2 Main 3 Alternate
Codec	string	Shoutcast compression type: aac-lc, aac-he or mp3 (default if property is not set)
Coding	decimal int	RSP stream encoder Forward Error Correction (FEC) setting: number of bytes out of 255 used for FEC instead of stream data. This determines how many bad bytes can be corrected out of a block of 255.
Content-type	string	override the content-type (audio/mp3, audio/aac, audio aacp) passed to the shoutcast server based on the codec selection with the value specified here.
CRC	decimal int	RSP stream encoder: If non-zero, and CRC32 checksum is added to all network packets; handy if the operating system does not use checksums on UDP packets.

Property key	Value type	Function
Custom-Codec	string	command-line string to run an external audio encoder program to process audio prior to streaming. Raw, 16 bit audio is passed via standard-in, and compressed audio packets are read from standard-out.
Genre	string	Genre name to set the corresponding mp3 file metatag/shoutcast tag to.
Hour	decimal int	Set to the hour (0-23) in which to start the recorder. Blank or not set for immediate recording. After a recorder is started, it will wait until the hour:minute:second specified to actually start recording audio.
icq	string	Shoutcast encoder: icq tag sent to server
Interleaving	decimal int	RSP stream encoder: how many network packets (1 to 84) are contained in a RSP interleaver block column. Interleaver bytes are arranged as 255 columns by (Interleaving x Payload) rows
irc	string	Shoutcast encoder: irc tag sent to server
Limit	decimal float	If set, the recorder/encoder will stop after this number of seconds of elapsed recording time.
MakePL	bool (0 or 1)	Set to 1 to enable shoutcast track posting or to create an file playlist (ARServer play list in file format) for recorders to go with the recording.
Minute	decimal int	Set to the minute (0-59) in which to start the recorder in the hour specified in the Hour property.
Mode	decimal int	0 Mono 1 Stereo 2 Forced joint stereo (lame/mp3) 3 Forced Normal stereo (lame/mp3)
MSGPeriod	decimal int	MountainChill.com custom property, see source code
Multicast	decimal int	For an RSP encoder sending to a multicast address, this is set to non-zero value this specifies the multicast distance (IP packet TTL field)
Name	string	Name of the file to record tofor recorders, stream name for shoutcast encoders. This is not a full path, just the name for the file (see the Path property below). If blank, the date/time will be used to create a file name.
PacketFEC	decimal int	RSP stream encoder: If non-zero, packet level Forward Error Correction is applied to all network packets. The network packets are all 256 bytes long. 255 - Payload bytes are used for FEC.
Password	string	Shoutcast encoder: password used to authorize the stream sent to the shoutcast server.
Path	string	File path to the folder in which to place the recorded file.

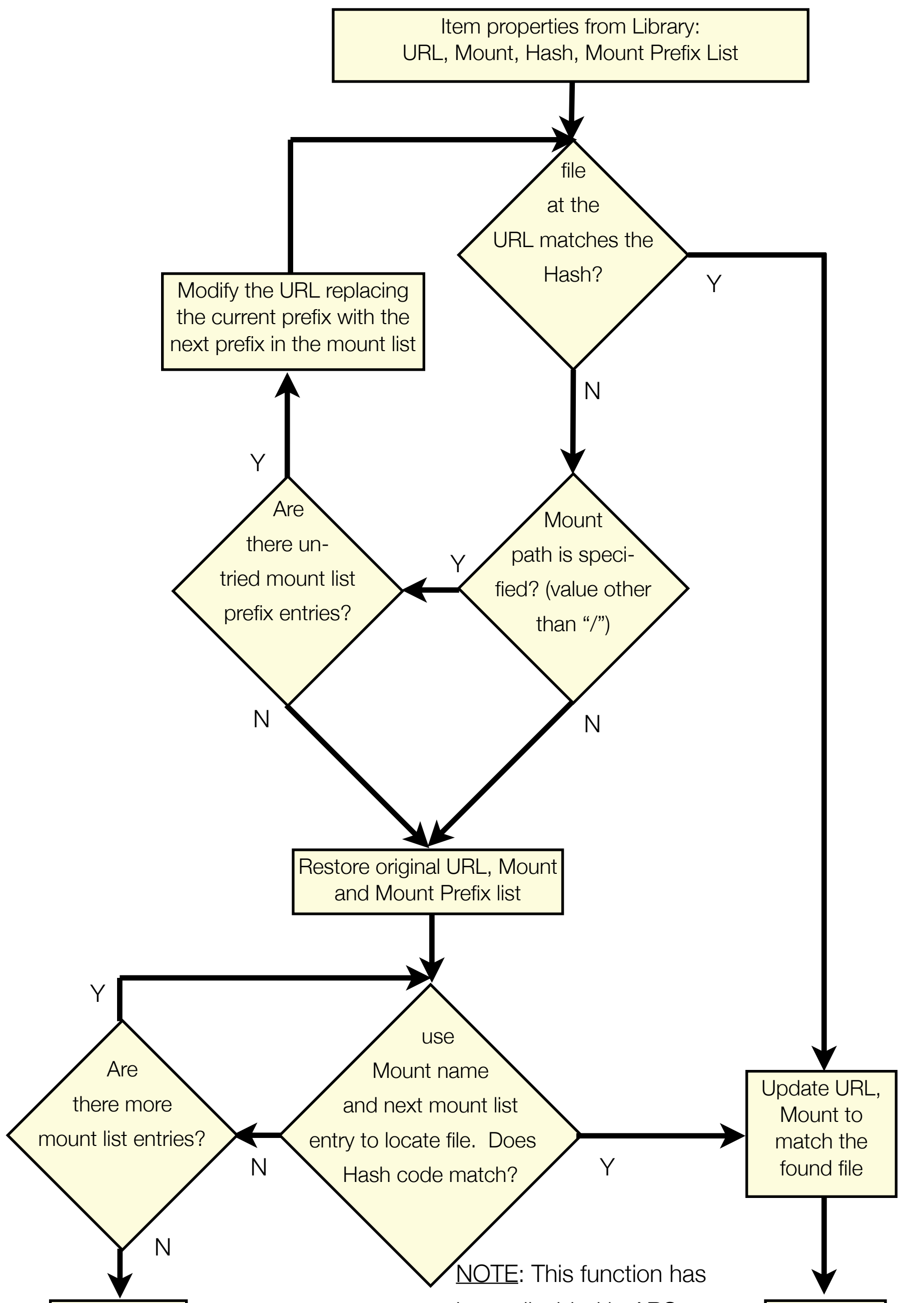
Property key	Value type	Function
Payload	decimal int	RSP stream encoder: how many bytes (16 to 256 in 16 byte steps) each network packet contains. Interleaver bytes are arranged as 255 columns by (Interleaving x Payload) rows
PrivateKeyFile	string	RSP encoder file path to a 2176 bit PEM formatted RSA private key to use for stream authentication.
Port	decimal int	Shoutcast/RSP encoder: IP port number to send the stream to. This is usually the base port number of the server plus 1 for shoutcast.
Public	bool (0 or 1)	Shoutcast encoder: set to 1 to make the stream public (listed at shoutcast.com)
Quality	decimal int	Compression quality: 0 is highest, 9 is lowest
SampleRate	decimal float	For uncompressed files, the output sample rate in samples per second.
SampleSize	decimal int	For uncompressed files, either 8 or 16 for the number of bits per sample.
Second	decimal int	Set to the second (0-59) in which to start the recorder in the hour and minute specified in the Hour and Minute property.
Server	string	Shoutcast encoder: IP address (name or doted quad format) of the shout-cast server to send the stream to.
Source	string	Audio source for the recorder/encoder: either an ARServer audio input name or “mixer” for a mixer output bus
Timeout	decimal int	Shoutcast encode: Sets the number of seconds after the shoutcast server stops acknowledging data we sent to it before it closes the connection. The encoder will then attempt to reconnect to the server every Timeout seconds until the encoder is stopped or it reconnects to the server. Set to 0, leave blank or don’t set to use the default value of 30 seconds.
Type	string	Recorder type: aiff, wave, m4a, lame/mp3, shoutcast source stream or rsp stream (see http://redmountainradio.com/rsp)
url	string	Shoutcast encoder: streams web page url tag sent to server

Task meta data

Property key	Value type	Function
Category	decimal int	The library category ID for a category to pick from for category Pick type tasks
Command	string	The ARServer command string for Command type tasks Shell command string for Execute type tasks
Date	bool (0 or 1)	Set to 1 if the pick is to search for a file that starts with todays date (YYYY-MM-DD) for folder Pick type tasks (first priority)
def_segout	decimal float	A default segout time to be passed on to the final item selected

Property key	Value type	Function
First	bool (0 or 1)	Set to 1 if the pick selects the first file in alphabetical order from the folder for folder Pick type tasks (fourth priority)
Folder	string	The file system path to a folder to pick an audio file from for folder Pick type tasks
Mode	string	For custom query Pick type tasks, set to: random - insert a randomly select item from the results into the playlist queue weighted - same as above, only weighting to prefer the first results first - insert the first result only into the queue all - insert all the results into the queue in the order they are returned
Modified	decimal int	If not 0, audio files must be newer than the specified number of hours to be selected for folder Pick type tasks.
NoModLimit	bool (0 or 1)	If set to 1, and Modified is not 0, and no audio files were found in the folder that met the Modified requirement, the selection process is repeated with out the Modified criteria for folder Pick type tasks.
Path	string	File system path to a file/script/application to open for Open type tasks
Query	string	URL encoded SQL query text to execute against the library for custom query Pick type tasks. The result from the final query (multiple queries can be executed via ";<LF>" separator) is expected to return the "toc" library table IDs of the items picked via the "ID" result name.
Random	decimal int	If not 0, the task will randomly pick an audio file from the folder which has not been played in at least the specifies number of hours for folder Pick type tasks (fifth priority)
Rerun	bool (0 or 1)	Set to 1 if the pick tries to rerun the last file played from this folder for folder Pick type tasks (second priority)
Sequencial	bool (0 or 1)	Set to 1 if the pick tries to select the next file in alphabetical order from the last file played from this folder for folder Pick type tasks (third priority)
Subtype	string	Specifies the type of task: Pick, Command, Open, Execute
Supress	string	For category Pick type tasks, specifies either Artist, Album, Name or Item last played dates to use for sorting results such that most recent last played items are least likely to be picked.

Appendix D Library file location algorithm


Item properties from Library:
URL, Mount, Hash, Mount Prefix List
file at the URL matches the Hash?
Y
N
Modify the URL replacing the current prefix with the next prefix in the mount list
Y
Are there un-tried mount list prefix entries?
Y
Mount path is speci-fied? (value other than "/")
N
N
Restore original URL, Mount and Mount Prefix list
Y
Are there more mount list entries?
N
use Mount name and next mount list entry to locate file. Does Hash code match?
Y
Update URL, Mount to match the found file
N
Failed
Success

NOTE: This function has been disabled in ARServer due to an change in inode search-ability in OS 10.6

Appendix E File hash algorithm

AudioRack Suite use a combination of Message-Digest algorithm 5 (MD5), file length and CRC32 based algorithm to compute a unique 64 digit hexadecimal ID for audio files in it's library. This allows AudioRack to identify a file that has been copied or moved as being the either the same file or a copy of a file in it's database. It also provides a double checking mechanism for locating moved or renamed file to ensure that the file it determines to be the file of interest is in fact that file.

It would be very inefficient to take the entire data contents of a file and run a Hash code computation on it. Instead AudioRack computes the Hash code from the MD5 algorithm of two concatenated block of 4096 byte from the file, followed by the 64 but file length, followed by the CRC32 checksums of three more 4096 byte blocks. The First MD5 block used is the 4096 byte block of the file that contains the byte one third into the file from the start. The second block is the block that contains the byte two third into the file from the start. The blocks are all aligned at 4096 byte boundaries from the start of the file, and the first block can be the same as the second block. For example, if a file is 6,000 bytes long, the first third byte location would be at location 2,000 from the start, the second third byte would be at location 4,000. Both locations are with in the first 4096 byte block, the the MD5 would be computed from the first block followed by the first block again. If the block is less than 4096 bytes long, then all the bytes that are available in that block are used.

The same process is used fro the CRC32 block, 1/4, 1/2 and 3/4 into the file.

The c language code used in ARServer to perform this calculation is shown below as two functions. GetFileMD5New generates the first 32 hex digits, and GetFileCRC generates the last 32 hex digits. The full Hash code is the concatenation of the two results.:

```
unsigned char GetFileMD5New(char *file, unsigned char *md5_str)
{
	FILE *fp;
	MD5_CTX md5;
	unsigned char block[8192];
	size_t len, b1, b2;
	int bSize;

	// get md5 hash of the 4K blocks 1/3 and 2/3 into the file
	if((fp = fopen(file, "r")) == NULL)
	  return false;
	// go to the end of the file
	fseek(fp, 0, SEEK_END);
	len = ftell(fp) / 3;
	b1 = (len / 4096) * 4096;
	b2 = (len / 2048) * 4096;
	fseek(fp, b1, SEEK_SET);
	bSize = fread(block, 1, 4096, fp);
	fseek(fp, b2, SEEK_SET);
	bSize = bSize + fread(block+bSize, 1, 4096, fp);
	fclose(fp);

	MD5_Init(&md5);
	MD5_Update(&md5, block, bSize);
	MD5_Final(md5_str, &md5);
	return true;
}
```

```
unsigned char GetFileCRC(char *file, unsigned char *result)
{
      FILE *fp;
      UInt32 crc_table[256];
      unsigned char blockA[4096], blockB[4096], blockC[4096];
      size_t fsize, len, b1, b2, b3;
      int sizeA, sizeB, sizeC;
      struct packing{
               UInt32         fsize;
               UInt32         crcA;
               UInt32         crcB;
               UInt32         crcC;
      };


      // get alternate hash of the 4K blocks 1/4, 1/2 and 3/4 into the file
    if((fp = fopen(file, "r")) == NULL)
        return false;
      // go to the end of the file
      fseek(fp, 0, SEEK_END);
      fsize = ftell(fp);
      len = fsize / 4;
      b1 = (len / 4096) * 4096;
      b2 = (len / 2048) * 4096;
      b3 = (3 * len / 4096) * 4096;
      fseek(fp, b1, SEEK_SET);
      sizeA = fread(blockA, 1, 4096, fp);
      fseek(fp, b2, SEEK_SET);
      sizeB = fread(blockB, 1, 4096, fp);
      fseek(fp, b3, SEEK_SET);
      sizeC = fread(blockC, 1, 4096, fp);
      fclose(fp);

      chksum_crc32gentab(crc_table);
      ((struct packing *)result)->crcA = EndianU32_NtoB(chksum_crc32(blockA, sizeA, crc_table));
      ((struct packing *)result)->crcB = EndianU32_NtoB(chksum_crc32(blockB, sizeB, crc_table));
      ((struct packing *)result)->crcC = EndianU32_NtoB(chksum_crc32(blockC, sizeC, crc_table));
      ((struct packing *)result)->fsize = EndianU32_NtoB(fsize & 0xffffffff);
      return true;
}
```

Appendix F Playlist storage

AudioRack Suite stores play lists as ordered collections of item meta data. When a play list is saved, the meta data available for each item is saved. When the play list is loaded, AudioRack suite will use as many of the meta data properties as it needs and are available to locate an item: This can be as little as a URL or ID property (from the library database) or a full set of properties. If only minimal meta data is available, there is a good chance that the item may not be locatable. For example, if only a URL is available, and the file has been moved, then AudioRack will not be able to locate it. If however, the item has mount information, and Hash properties, then other means can be use to find it. If a Database ID is included, even more options are available for item location. Name, Artist and Album, as a last resort, will be used against the library if available and no other location technique has found the item.

AudioRack has two formats for play list storage: File and Library.

A file play list is a text file with a .fpl file name extension. The file contains a header, then an ordered list of items with all their properties. Each item and the header is composed of a line of text for each property the item contains: The property name followed by a tab character followed by the property value and terminated with a Line Feed charter. The order of the properties with in an item is not important. The last property for an item is followed by a blank line - just a Line-Feed character. This marks the end of the item. The next line will contain the first property of the next item in the list, if any. The first item is a header, which contains properties for the play list as a whole. The second item then would be the first item in the play list. See the following example.

File Contents		Section Description
Type	filepl	Required Header: First entry MUST be “Type<tab>filepl”, all other
Name	smallList	entries are required but the order is not important. Fingerprint is
Duration	542.54	the fingerprint of the database that subsequent item ID are refer-
Fingerprint	596533461	enced to. The Duration is the total length, in seconds, of all the
Revision	1.0	items in the playlist.
		[blank line] end of header item
Name	My Sunset	First list Item and it’s meta data (See appendix C)
Mount	/	Offset is the time in seconds, from the start of the play list, when
ID	2512	this item will play.
SegOut	307.27	
Hash	3cc73746e4109c727b910a9d7e39f027...	
URL	file://localhost/Users/ethan/Desktop/The%20A...	
FileID	2046922	
Type	file	
Album	The Art of Chill 2	
Duration	312.27	
Artist	Chris Coco	
Offset	0	
		[blank line] end of item
Name	Deliver Me.mp3	Second list Item and it’s meta data... just as above
SegOut	235.27	
Mount	/	
Hash	4c614360da93c0a041b22e537de151eb...	
URL	file://localhost/Users/ethan/Desktop/Deliver%20...	
FileID	2113605	
Type	file	
Album	Pure Moods III	
Duration	240.27	
Artist	Sarah Brightman	
Offset	307.27	
		[blank line] end of item. No further data: End of list...

A library play list is similar in concept to the file playlist except that it is stored in the library database in the “Playlist” table. Every library playlist has a ID from the libraries table of contents (the “toc” table) which contains Name and duration properties. Note that all items in the library, files, tasks playlists, etc, all have entries in the library table of contents table. This table of contents entry is similar to the header in the file type play list. The actual meta data for items in the list are stored in the “Playlist” table. A “Position” column in the table is used to assign an order to items, grouping all the properties of a common item with a position index number in the list, starting with zero for the first item. The first item’s properties MUST have position values of zero and position values that follow must be continuous - you can’t skip numbers. See the following example.

ID	Position	Property	Value
11437	0	Artist	Various Artists
11437	0	Duration	312.27
11437	0	Album	The Art of Chill 2
11437	0	Type	file

ID	Position	Property	Value
11437	0	FileID	2046922
11437	0	ID	2512
11437	0	URL	file://localhost/Users/ethan/Desktop/The%20Art%20of%20Chill%202_Various%20Artists_4_My%20Sunset.mp3
11437	0	SegOut	307.27
11437	0	Hash	3cc73746e4109c727b910a9d7e39f0273cc73746e4109c727b910a9d7e39f027
11437	0	Mount	/
11437	0	Name	My Sunset - Chris Coco
11437	1	Artist	Sarah Brightman
11437	1	Duration	240.27
11437	1	Album	Pure Moods III
11437	1	Type	file
11437	1	FileID	2113605
11437	1	URL	file://localhost/Users/ethan/Desktop/Deliver%20Me.mp3
11437	1	Name	Deliver Me.mp3
11437	1	SegOut	235.27
11437	1	Mount	/
11437	1	Hash	4c614360da93c0a041b22e537de151eb4c614360da93c0a041b22e537de151eb

Note that there is no offset property in library playlist. The Offset property is used only for file playlists when an audio file is being played by AudioRack which has a file play list in the same directory with the same name as the file with .fpl appended to the end. When such a file (associated play list) is found, ARServer will make log entries as the audio file plays for items in the play list with an offset time correspond to the audio playback time that has just been passed. For example, if an item in an associated play list file has an Offset value of 200, then ARServer will make a log entry with the Name, Artist, Album, etc values of that item when it has played just past the 200 seconds mark in the audio file.

Appendix G MODBUS Relay Interface

AudioRack Server uses the MODBUS TCP protocol in order to trigger the starting and stopping of equipment such as turntables and CD players associated with an audio input or a ARServer mute bus, and to trigger On-Air lights and telephone ringer mutes commonly used with microphone inputs. MODBUS can also be used to cause external relay events to trigger actions in ARServer. MODBUS TCP is an open protocol. You can find more information about the protocol on the web, as well as finding MODBUS TCP relay input and output hardware available from numerous vendors.

There are two elements in the modbus interface: Trigger scripts and modbus commands. Trigger scripts are ARServer configuration files containing one or mode ARServer commands that are executed by ARServer when an event occurs. Modbus commands are ARServer commands that either control external relays using the MODBUS TCP protocol, or create a background task that poll external modbus relays and run Trigger scripts when the polled relay changes state.

Trigger scripts must all be located in the same directory, which is specified by the ARServer `file_trigger_dir` setting. This setting is not set by default, so you must set it before trigger scripts will work. If ARServer is running in daemon mode, then it is recommended that the directory /etc/arserver/triggers be created and used for this purpose. Use the ARServer command:

set file_trigger_dir /etc/arserver/triggers
saveset

to set and save this as the directory to use for event triggers after you have created the directory and set it's access permissions so ARServer can read files in it.

Outgoing Triggers - ARServer triggers a relay output

1. A Trigger script is run when an input group or mute group starts or stops playing. The trigger script has the same name as the input group or muteA, muteB, muteC, and Cue for mute groups that caused the event. The trigger script has either .start or .stop appended to it's name to indicate if it should be run as a start or stop event.

2. The trigger script contains a modbusset or modbusclear command to cause the desired relay to be set or cleared. You could have other ARServer commands executed as well.

Incoming Triggers - A relay input triggers an ARServer action

1. The modbuspoll command is executed to start background polling of the specified modbus relay, and associated that relay with a specified trigger script. Modbuspoll commands are usually added to the ARServer startup configuration script so they run automatically when ARServer is started.

2. When one of the polled relays changes state, the associated trigger script with either .on or .off, depending on the new state of the relay, is run. This script can contain any number of ARServer commands you want to be executed when the relay changes states.

Here is a complex working example of both relay inputs and output:

The ars_startup.conf file contains the following lines to set initial output relay states and start the polling process for two input relays.

```
modbusclear 192.168.15.10 01 0010
modbuspoll 2 192.168.15.10 01 0000 EAS-Request
modbuspoll 2 192.168.15.10 01 0001 EAS-PTT
```

The ARServer setting file_trigger_dir had already been set to /etc/arserver/triggers and saved. This directory contains the following files performing the functions described. All trigger scripts must end with a new-line character or the last command will not be executed. Note: EAS is the Emergency Alert System. This is a piece of equipment that must be able to break into programming to air emergency messages, either immediately or waiting until the current item finishes depending on the importance of the emergency message.

Name: EAS-PTT.on

Function: High priority EAS message request: break into programming right now. Assumes there is a modbus polling task (see modbuspoll command) checking on the status of a relay mapped to the EAS-PTT script name, when the polled relay input goes high (request termination), The currently playing item, which should be the EAS input group, is faded and the next item is the playlist queue is started and audio processing in the FM transmitter output group chain is turned back on.

Contents:

```
segnow
fxbypass 00000002 0
fxbypass 01000002 0
```

Name: EAS-PTT.off

Function: EAS message is done. Assumes there is a modbus polling task (see modbuspoll command) checking on the status of a relay mapped to the EAS-PTT script name, when the polled relay input goes low (active request), automation is turned on just incase it's off, the EAS audio input group is load next in the playlist queue IF IT IS NOT ALREADY LOADED, TargetTime and Priority of the item are set so it plays immediately, fading the previously playing item if needed, and bypassing audio processing in the FM transmitter output group chain.

Contents:

```
autoon
uadd -2 input:///EAS
setmeta -1 TargetTime -1
setmeta -1 Priority 10
fxbypass 00000002 1
fxbypass 01000002 1
```

Name: EAS-Request.off
Function: Low priority EAS message: requests EAS to play next. Assumes there is a modbus polling task (see mod-buspoll command) checking on the status of a relay mapped to the EAS-Request script name, when the polled relay input goes low (active request), the EAS audio input group is load next in the playlist queue, TargetTime and Priority of the item are set so it stays next in the list, and automation is turned on, just incase it's off.
Contents:

```
add -2 input:///EAS
setmeta -1 TargetTime -1
setmeta -1 Priority 9
autoon
```

Name: EAS.start
Function: Signal back to EAS that it's on the air. When the input group named EAS has started playing, modbus relay 16 (10 hex) of the relay controller 01 at address 192.168.15.10 is turned on to signal to the EAS system that it is live and can now run it's message.
Contents:

```
modbusset 192.168.15.10 01 0010
```

Name: EAS.stop
Function: Signal back to EAS that it's off the air. When the input group named EAS stopes playing, modbus relay 16 (10 hex) of the relay controller 01 at address 192.168.15.10 is turned off to signal to the EAS system that it is no longer on the air.
Contents:

```
modbusclear 192.168.15.10 01 0010
```

Name: TT.start
Function: Start the turntable. When the TT input group (used by a turntable) starts playing, modbus relay 19 (13 hex) of the relay controller 01 at address 192.168.15.10 is turned on to signal to the Turntable to start running.
Contents:

```
modbusset 192.168.15.10 01 0013
```

Name: TT.stop
Function: Stop the turntable. When the TT input group (used by a turntable) stops playing, modbus relay 19 (13 hex) of the relay controller 01 at address 192.168.15.10 is turned off to signal to the Turntable to stop running.
Contents:

```
modbusclear 192.168.15.10 01 0013
```

Name: Start studio mute (all studio microphones are assigned to mute group A). muteA.start
Function: When the muteA ARServer mute bus is active (any input associated with this mute bus is playing) modbus relay 18 (12 hex) of the relay controller 01 at address 192.168.15.10 is turned on to light the studio On-Air light and mute the studio phone ringer.
Contents:

```
modbusset 192.168.15.10 01 0012
```

Name: muteA.stop
Function: Stop studio mute (all studio microphones are assigned to mute group A). When the muteA ARServer mute bus is inactive (no input associated with this mute bus is playing) modbus relay 18 (12 hex) of the relay controller 01 at address 192.168.15.10 is turned off to extinguish the studio On-Air light and un-mute the studio phone ringer.
Contents:

```
modbusclear 192.168.15.10 01 0012
```

Appendix H Resilient Streaming Protocol (RSP)

AudioRack Server supports the Resilient Streaming Protocol (RSP) as an alternative to shoutcast streaming for delivery of an audio stream in any of the formats currently supported by ARServer (lame/mp3, aac-lc, aac-he, etc). This is a new streaming protocol developed specifically for my use in reliably distributing audio to mountanchill.com streaming servers and the KRKQ FM transmitter. ARServer's RSP stream encoder should be considered the reference encoder for the protocol. What follows is an overview of the inner working of the protocol, followed by some information on setting up a stream encoder in AudioRack. As I continue to develop this protocol, expect ARServer to gain the ability to play RSP streams and the example relay server to evolve to server shoutcast clients (relaying) along with the currently supported RSP client relaying. My goal is to have ARserver encode an RSP stream sending to multiple RSP relays host serving both shoutcast and RSP clients with a single listener reporting system. Up to date information on the protocol and code examples of a client player, a shoutcast source bridge and a relay server can be found at http://redmountainradio.com/rsp.

The Protocol

1.0 OVERVIEW

The Resilient Streaming Protocol is an Internet protocol built on top of the Unreliable Datagram protocol (UDP) layer for transporting a unidirectional, high reliability, multimedia data stream and a low bit rate meta data stream where latency is not a concern. The protocol is designed for one-way transmission, making it well suited for multicast distribution, but it also includes receiver reporting and requesting mechanism to allow for connection setup and teardown in a unicast distribution environment, and for listenership tracking in both multicast and unicast networks.

UDP packets are by nature unreliable. Packet may arrive out of order, or not arrive at all. To improve the reliability of the data reception with out creating the need for a bidirectional connection (such as the packet resend request mechanisms used in many UDP multimedia protocols, and by all TCP protocols), rsp makes use of Forward Error Correction (FEC) and data interleaving over time. This allows the receiver to piece together and correct blocks of data with out needing to contact the sender.

The protocol uses out-of-band channels (typically a jSON text file) to communicate connection settings (ports and addresses, etc), content format, and rsp protocol specific settings such as interleaver and FEC properties. The sender can then tailor the FEC overhead (increase in coded data rate) and interleaved parameters (latency to fill an interleaver block) for the packet losses they are willing to tolerate before the receiver can no longer piece together an error free block. RSP make no assumptions about the format of the data being sent; ADTS, for example, is a common framing mechanism used for AAC audio packets. It is the responsibility of out-of-band signaling method to make sure the receiver knows how to handle the data transported.

Because rsp is a connectionless protocol, measures need to be taken to prevent malicious transmission of packets by those who might want to "hijack" the stream. RSP allows an RSA private key encrypted authentication packet to be sent to receivers which contains 8 bit check sums for all the packets about to be sent from a given interleaver block. The receivers can use a public key (again sent out-of-band, typically part of the jSON file) to decrypt this packet and then verify the data checksums prior to inserting a packets data into the interleaver. While an 8 bit checksum is very weak, it is highly unlikely that a malicious sender could get more than one malicious packet into a block. In this case, the rsp FEC mechanism will simply correct the errors that the slipped in packet creates in the block.

To further improve reliability, rsp allows redundant reception of network packets: a receiver can handle receiving the same packet more than once.This allows a sender to transmit packets to listeners using multiple internet connections at the same time. The receiver will simply insert the packet it receive first into the interleaver, and ignore subsequent packets addressed to the same column. Sender UDP packet replication is typically performed at the senders router. The protocol allows for handling duplicate reception, but does not itself implement the replication. It should be noted that duplicate packet transmission does increase the total data rate at both the sender side, split across multiple network connections, and at the receiver side who will most likely be receiving duplicate packets through the same network connection.

To further improve robustness at the network level, packets can optionally have packet level forward error correction and/or a packet level CRC32 check sum added.

1.1 INTERLEAVER

Interleaving is important to the design of a protocol of this nature in order to spread chunks of loosed data (packets) across time. Forward error correction algorithms can correct up to a certain percentage of bad bytes of data in a block. However, it still needs some data in the block to work with. If there is a burst of packet loss, a large part of or an entire block of data could be lost, preventing recovery of the data. If the bytes from packets are spread across time, then a sequential burst of packet loss will be spread across time as a few bad bytes here and there, which the forward error correction algorithm can easily correct.

The rsp interleaver consists of N blocks, where each block stores bytes in 255 columns of pay-load size bytes. The row size must be a multiple of 16 bytes and less than 256. With in this structure, each column contains the time-interleaved pay-load content of a network packet carrying multimedia data. Rows are written in or read out to access the time-linear data. For example, a sender would write data into interleaved rows sequentially, and once a block is full, read the network payout out the columns sequentially for network transmission. The process is reversed on the receiving size.

An additional "interleaving number" is specified to allow further spreading for network pay-load across multiple blocks grouped into logical blocks. For example, if the "interleaving number" is set to 3, then a sender, after filling the rows of 3 blocks, would have filled a complete logical block and could then read column data out as follows:

Block	Column
0	0
1	0
2	0
0	1
1	1
2	1
0	2
1	2
2	2
0	3
1	3
...	
0	254
1	254
2	254
3	0
4	0
5	0
...	

If the interleaving number is 1, then there is only one block per logical block:

Block	Column
0	0
0	1
0	2
0	3
...	
0	254
1	0
1	1
...	

It is advisable for a receiver to fill 1.5 logical blocks before starting to read data out of the rows of the first full logical block to allow for network jitter. The interleaver is always set to roll the block numbers back over to 0 after interleaving number * 3 blocks to allow for twice the 1.5 logical block threshold. Be aware that this will introduce a receiver latency (delay) equal to 1.5 to 2 logical blocks, depending on how far into a block the stream was when the receiver started listening. If, for example, a stream is encoded at 128 kb/s, and the RSP parameters are set such that a logical block carries 40,448 bytes and there for 323,584 bits of stream data, then each block carries 316/128 = 2.47 seconds of stream data. The receiver then could have between 3.71 and 4.94 seconds delay. The transmitting end will always have a fixed one logical block delay since it can't start sending interleaver data until a full block has been filled. So in this case, there is almost 7 seconds of delay from end to end.

The first byte in each row of interleaved data is reserved for meta data. Metadata is transmitted as a jSON formatted text stream, usually with out printability formatting such as TAB, LF and CR characters. A completed jSON structure is always terminated by a NULL character. When there is no new meta data to send, NULLS are used to fill the row meta data bytes. The receiver can then use received NULL characters in the meta data stream as tokens for starting to parse a new jSON formatted data block.

In implementing the reading and writing of data to and from the interleaver, for both transmitters and receivers, it is advisable to pace the reading out of data from a full block to match the write in rate of data to a filling block. This allows the data to be sent and received in a steady fashion rather than sending nothing for a long period of time as a block fills, followed by a burst to high rate data to then send the whole block at once.

1.2 FORWARD ERROR CORRECTION

Reed-Solomon (RS) forward error correction encoding is always applied to all rows written into the interleaved by a sender, and all rows read from the interleaver by a receiver must be decoded accordingly. An encoded row is always 255 bytes long. The number of bytes required to generate that row is selectable by the sender to allow the sender to choose how much throughput to trade off for in exchange for the maximum number of erroneous bytes that can be corrected. The remaining number of bytes in the 255 byte row are parity bytes generated the the RS encoding process. The RS algorithm can correct up to parity bytes number of errors per row, assuming it knows which bytes in the rows are bad/missing. The RSP protocol uses packet column and block addresses to note which interleaver columns have been filled and conversely which are missing, so it can identify the known bad bytes in a row. If an erroneous byte works it way into the interleaved with out the interleave's knowledge, the FEC can still correct the errors, only two parity bytes are requires for each error byte for the correction to be successful.

The RSP protocol has the option of also apply additional RS encoding to each network packet before sending. If this option is enabled, then the second through nth byte each sent network packets is also RS encoded, such that the number of parity bytes is equal to 255 minus the original packet size - 1 prior to encoding.

1.3 CYCLIC REDUNDANCY CHECK (CRC)

Some RSP packets can have a CRC32 checksum appended to the packet. This is useful to ensure the integrity of the packet if the packet is being generated on an operating system that does not support UDP packet checksums. It is better for RSP to receive no packet at all then to receive a packet with bad data, due to the nature of RS. This option help ensure this is the case where the sending operating systems doesn't handle this for you.

CRC checksums are mandatory for authentication packets, as a mechanism to validate the encrypted data.

1.4 AUTHENTICATION

To provide some level of source authentication to a receiver of a RSP stream, periodic authentication packets are transmitter which allow the receiver to discard packets it receives that may be coming from a malicious source. Public/private key encryption of all RSP packets can be burdensome to the receiver in terms of processing resource, so instead, RSP sends one encrypted authentication packet just ahead of sending data for an interleaver block. The authentication packet contains a payload of one interleaved block number and 255 8 bit check sums, that correspond to the 8 bit check sums for each column in the interleaved for which the authentication packet represents. Note that the checksum is simply the 2s compliment of the 8 bit sum of all the bytes in the interleaved column. The receiver can then validate unsecured/unencrypted payload received packets addressed to the interleaved block and column agains the check sums that were transmitter securely. Any packets that don't match the checksum are thrown out. If packets are received prior to receiving authentication packets, then all payload packets are excepted, making authentication packets both optional and causing no harm if they are lost in transmission.

Authentication packets always have a CRC32 checksum calculated on the contents, including the packet header. Then, all but the first header byte are RSA encrypted with a 2176 bit (272 byte) private key using PKCS1 padding. This results in a 273 byte packet, at the end of which the pre-encryption 4 byte CRC32 check sum is appended to create a 277 byte packet. The receiver then notes the CRC32 bytes at the end of the packet, decrypts the 272 bytes after the header byte using a public key it received out-of-band (again, most likely via a jSON file), and checks the unencrypted result including the packet header byte again the CRC32 it received. If the CRC32 results match, then the checksums are stored to checking subsequently received packets.

1.5 RECEIVER REPORT/REQUEST/REPLY MECHANISM

To facilitate both statistical data collection from all receivers (at the receivers option) and stream initiation and termination for relay based unicast listeners, RSP includes a mechanism for receivers to send messages about themselves to a relay server, which replicates RSP packet streams to many uniccast listeners, or a statistics collection server, or a combination of the two. Receiver Report/Request/Reply (RR) packets carry a NULL terminated, jSON encoded string payload, usually, although not required, in non-printable format (no tabs, white space and line formatting) for compactness. Packet payload size is specified in 16 byte blocks, so the string payload length is always rounded up to the next 16 byte size, and padded whit NULLs.

A RSP relay server can also send messages to a listener it is relaying to via this packet type.

1.6 META DATA STREAM

As previously mentioned in the interleaved section, the first byte of each interleaver row is reserved for caring a meta data stream to go along with the media data in the stream in real time. Meta data bytes are set to NULL (zero) if there is no meta data to be sent at the time. If there is meta data to be sent, the bytes are filled with a jSON string containing the meta data, such as currently playing track information, or messages to listeners. A NULL byte is always added to the end of the jSON string so the receiver knows when it has received a complete jSON formatted string and it can proceed to pares it. Note that the jSON string is typically not formatted for printability (no white space or line characters) to save space, however, this is not a requirement.

Here are some jSON string examples:

New track information jSON string:

```
{"item":{"ID":16435,"Name":"A Gentle Dissolve","Artist":"Thievery Corporation","Album":"The Cosmic
Game","AR":{"FP":3546701,"LocID":3,"ID":54578,"Artist":12747,"Album":8945}}}
```

item:	Standard Track information	
	ID:	A unique ID number for this meta data entry; if it is sent more than once, the receiver can ignore subsequent entries.
	Artist:	Artist name
	Album:	Album name

Name:	Song title.
AR:	An example of sender specific custom information, most likely to be ignored by typical listeners, but which may be useful to others. In this case, it's custom data of use to the AudioRack Suite radio automation system.

FP:	Finger print number: a unique identifier for the AudioRack database the following information applies to.
LocID:	AudioRack database location ID number.
Artist:	Database Artist ID number.
Album:	Database Album ID number.
ID:	Song's database ID number.

All sorts of custom and application specific data is possible. The above is just an example.

Message to listeners jSON string:
{"message":"Test message"}

2.0 RSP PACKET STRUCTURE

```
GENERAL FORM

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|ID |C R| Size  | Block (opt)   | Column (opt)  |                          Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                      |
|                        Variable length Payload                       | Data being carried
|                                                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Optional CRC32  checksum of all previous bytes in the packet      | Optional Checksum
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

NO PACKET LEVEL FEC

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|ID |C 0| Size  | Block (opt)   | Column (opt)  |               Header, no packet level FEC applied
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                      |
|                         Variable length Payload                      | Data being carried
|                                                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Optional CRC32  checksum of all previous bytes in the packet      | Optional Checksum
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

WITH PACKET LEVEL FEC

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+
|ID |C 1| Size  |                                             Header with packet level FEC applied
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                      | Data being carried, 2
|                        255 Bytes of RS encoded data                  | through Nth byte of
|                                                                      | original packet encoded
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ using RS encoding with
                                                                          255-(N-1) parity bytes
```

2.1 ID FIELD:

0 Payload packet, 3 byte header, interleaver column data from the specified interleaver block and column. This packet includes both Block and Column number bytes

1 Authentication packet, 2 byte header, 255 interleaver column 8 bit checksums for the specified interleaved block, RSA private key encrypted. This packet includes a Block number byte.

2 Receiver Report/Request packet, 1 byte header, jSON formated information about a receiver.

3 Reserved

2.2 FLAGS:

C Packet included a CRC32 checksum appended to the end. The checksum includes all packet bytes ahead of the checksum, with the exception of authentication packets, where the CRC is calculated for the entire contents of the packet PRIOR to RSA encryption.

NOTE: Authentication packets should always have the C flag set.

R The packet is Reed-solomon Forward Error correction encoded. All bytes, except the first byte are RS encoded to produce 255 byte result. The packet is always 256 bytes long regardless of how much data it was carrying prior to encoding. The RS encoding process is set to fill out the extra bytes with parity codes. The Size field of the first byte specifies the number of payload bytes carried prior to encoding and after decoding, excluding header and optional CRC bytes. To decode the packet, it is required to know the number of parity bytes added by the encoding process. This can be extracted from the size field as follows: Parity bytes added = 256 - (Payload-size + (header size and CRC)).

NOTE: if both the C and R flags are set, the CRC32 check sum is added to the packet on transmission befor the packet had been RS encoded.

NOTE: Authentication packets can never have the R flag set.

2.3 SIZE FIELD:

Number of raw payload bytes, in 16 byte steps, less first 16 bytes, carried in the packet. On packet transmission, this count applies to the number of bytes begin carried in the packet before any RS encoding, and not counting any CRC check sum additions and all header and interleaved address bytes, if any. On reception, this is the number of actual data bytes you will get out of the packet after all the headers, CRC checks and RS coding have been removed.

Example: 96 bytes payload, Size = (96 / 16) - 1

3.0 PACKET TYPE DETAILS, all shown with out packet level FEC.

3.1 PAYLOAD PACKET

```
 0                   1                   2                         3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0|C R| Size  |     Block     |    Column     |                           Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                       | Data being carried,
|                       Interleaver Column data                         | 16 to 256 bytes
|                                                                       | in 16 byte steps
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Optional CRC32  checksum of all previous bytes in the packet        | Optional Checksum
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Block = 0 to 255 decimal
```

```
Column = 0 to 254 decimal

If Column = 255 decimal, then the transmitter is restarting a stream and all receivers should
reset/clear their interleaves.  Transmission should always start out with sending several reset
packets of this type.  Payload data is ignored in this case.
```

3.2 AUTHENTICATION PACKET

```
Sent/received over the network:
 0                   1                   2                         3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+
|1 0|1 0|1 1 1 1|                                                          Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                      | Data being carried,
| 2176 bit Private key RSA Encrypted interleaver row checksums         | 256 bytes encrypted
|                                                                      | as 272 bytes
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   checksum of all bytes in the packet prior to encryption            | Required Checksum, from
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ bytes 1 to 257 of the
                                                                         packet befor encryption

After decryption by the receiver:
 0                   1                   2                         3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+
|1 0|1 0|1 1 1 1|                                                          Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Block     |                                                      |
+-+-+-+-+-+-+-+-+    255 8 bit row check sums                          | 256 bytes decrypted bytes
|                                                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         16 bytes                                     | Ignore
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  checksum of all bytes up to and including 255 8 bit checksums       | Required Checksum from
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ bytes 1 through 257
```

3.3 RECEIVER REPORT/REQUEST PACKET

```
 0                   1                   2                         3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+
|0 1|C R| Size  |                                                          Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                      | Data being carried,
|            jSON string, NULL terminated, NULL padded                 | 16 to 256 bytes
|                                                                      | in 16 byte steps
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Optional CRC32  checksum of all previous bytes in the packet       | Optional Checksum
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Base receiver report/request jSON string:
{"Client":"simpleRSPPlayer","Stream":"Test Stream","IP4":{"Address":"192.168.0.4","Port":5075,"Multicast":"","Relay":True}}

Client: Name of client program.

Stream: Stream reference string for servers handling multiple streams.

IP4: Internet v4 connection data

Address:	The client's address it is listening for packets on, as perceived by the client (can be a NAT address for example)
Port:	The client's port it is listening for packets on, as perceived by the client (can be a NATed port for example)
Multicast:	Multicast group address if this client is listening via multicast
Via:	For reports being passed from one relay server to another, this is the originating relay server's address. If this property is present, the Address and Port properties are assumed to be the client address and port as perceived by the relay server.
Relay:	True if this listener is a unicast listener requesting relay service from this server. This property must be false or not present if the Via property exists.

IP6: Same as IP4, only Internet v6 address formats, etc.

Receiver Report example jSON string:
{"Client":"simpleRSPPlayer","Stream":"Test Stream","IP4":{"Address":"192.168.0.4","Port":5075,"Multicast":"","Relay":True},"Report":{"Fixed":0.003349,"Failed":0,"BadPkt":0,"DupPkt":49.981558}}

Report: Packet reception report data, averaged over a full interleaved group.

Fixed:	Number of bytes per interleaver row that the FEC corrected.
Failed:	Percent of interleaved rows that had more errors that the FEC could correct.
BadPkt:	Percent of packets received that were bad: failed CRC, authenticated check sum, or where not formatted correctly.
DupPkt:	Percent of packets received that were duplicates, or -100 for break in stream reception.

Receiver Start Request packet example jSON string:
{"Client":"simpleRSPPlayer","Stream":"Test Stream","start:true,"IP4":{"Address":"192.168.0.4","Port":5075,"Multicast":"","Relay":True}}

start: true/false - set to true if this is a start streaming request to a relay server.

Receiver Stop request packet example jSON string:
{"Client":"simpleRSPPlayer","stop":true,"IP4":{"Address":"192.168.0.4","Port":5075,"Multicast":"","Relay":True}}

stop: true/false - set to true if this is a stop streaming request to a relay server.

Relay server message to listener packet example jSON string:
{"Server":"rspRelayerServer","error":"Server full"}

4.0 APPENDIX

Example jSON file advertising a unicast stream via a relay server:

```
{
	"rspStream": {
		"Name": "Test Stream",
            "Format": {
                "FEC": 112,
                "Payload": 192,
                "Interleave": 16,
                "RS": false,
                "CRC": false,
                "RSAPublicKey": "-----BEGIN PUBLIC 
KEY-----\nMIIBMjANBgkqhkiG9w0BAQEFAAOCAR8AMIIBGgKCAREA7y5MwQ51QDiXSfZAXfkx\ng1a8T3Khl0GfsZ6yuCUT2tngp8
xLquS01hFMS+c2oR6mTjsaOarMEpupKchHjljc\nGwyBk+FtUc+8KpWe6+3mSBSddBYgvZzvKwmAPjQga0r4oLkTuP2qQMKo1zc
6rlCv\nfcnbxMdXOUDDO6VGu67mqTkRs6NCjAb2OStoqKVTS3CpaLy+7IOEhUyQVtzoWpoM\nTmnKaa0TM6qTFz4UAOi5Al6lGF
G9JbrxwnnWQVOHcsIgLxPRpsDfPRaJwD83HJ34\ni3LoQ5XNcgcXVT9CDcdfZT7z2QYBc6IHRdL/y3/zIaCMWTBmNRa7J7zSiuK8
FVm3\nfSy3a3XW/f296rdm/Uvg3pcCAwEAAQ==\n-----END PUBLIC KEY-----"
                },
            "Content": {
                "Type": "audio/aac",
                "SampleRate": 32000,
                "kBitRate": 128,
                "Channels": 2
            },
            "IP4": {
                "MulticastGroup": "",
                "Port": 0,
                "ReportHost": "rsp.redmountainradio.com",
                "ReportPort": 5075,
                "ReportPeriod": 20
            }
        }
}
```

Notes:

Name, if set, is passed along in receiver request reports, allowing it to be used as a stream reference when a server is hosting multiple streams.

FEC is configured for 112 Reed-Solomon parity bytes for each 143 data bytes (255 - 112) to produce a total of 255 network bytes. The first data byte is always from the meta data stream. This format will be capable of correcting 112 / 255 = 43.9% lost packets over the intreleaver period.

The interleaver is configured for an "interleaving number" of 16.

Each interleaver column and the packet payload is 192 bytes long.

From the above settings:
 Each interleaver block group carries 192 * 142 (FEC setting less one byte per row for meta data) * 16 bytes = 436,224 bytes. At 128 kb/s (16,384 bytes/s), the interleaver group is 26.6 seconds long. 43.9% packet loss over this period can be tolerated with out error.

No additional Reed-Solomon encoding is performed on the network packets.

No CRC checksum is generated for network packets.

Authentication packets will be sent, encrypted with the provided public RSA key. Note the \n escaped codes in the public key. jSON requires that this be a single line of text in quotes with escaped LF and/or CR characters.

This is an IPv4 unicast network stream. Connection requests and receiver reports are to be sent to rsp.redmountainradio.com on UPD port 5075. Receiver reports are to be sent every 20 seconds. If this host is also an RSP relay server, then when the first start request packet is received, the server at rsp.redmountainradio.com will begin sending replica rsp packets to the address and port number the report was received from until a request is received to stop, or a report has not been receive after some time-out period.

The rsp decoded data stream is a 128 kib/s stereo ADTS aac audio format stream at a sample rate of 32 ksps.

If "Port": was non-zero, then this would be direct, preset broadcast from the sender (who ever that would be) to this particular receiver with out the report host "relaying" replica packets from the source to the receiver. The receiver can still send receiver reports to the specified ReportHost.

If the "Multicast" group is specified, then the receiver would be expected to join the specified multicast group, and again, the report host will not "relaying" packets to the receiver, as it will be getting them from the multicast group. The receiver is still expected to send receiver reports to the specified ReportHost.

Using the AudioRack RSP encoder

Configuring a stream encoder in ARStudio is very similar to configuring a shoutcast encoder. There is a graphical user interface available for all the settings required in ARStudio. Note that this is ARServer's RSP encoder is NOT a relay server: you can configure the encoder to send RSP stream packets to a number of destination addresses entered in the Addresses field (comma separated), but these destinations are static... when the encoder is running, packets will be sent to even if there is no RSP client running at those destinations. The intent is to send packets to static addresses at which an RSP relay server (to which clients connect when they request a stream be replicated to their address) or a shoutcast bridge is running. If your network supports multicast, you can send packets to a multicast group address as well. The other encoder settings, such as Interleaving, Packet Payload size, and FEC Coding are explained in detail above.

Two related tasks that may require explanation are the creation of the jSON file used to advertise the stream and the creation of RSA key pairs if you want to use stream authentication to prevent your stream from being hijacked. The latter can be achieved using the following two openssl commands in the terminal application:

```
openssl genrsa -out rsa.private 2176 <return>
openssl rsa -in rsa.private -out rsa.public -pubout -outform PEM <return>
```

You will now have two files rsp.public and rsa.private in the working directory your terminal session was in when you ran the commands. The you will want to edit rsp.public file text to convert all the new-lines in the file into the new line escape sequence "\n" so the file is one continuous line of text, ready to be pasted into the jSON advertising file. The rsp.private file should be moved into a director that ARServer has permission to read from and no other users can see. The file itself must also be set for read permission from the user account that ARServer runs under, with no other users having read or write access. If you are running ARServer in daemon mode, then it runs under a user simply named "arserve". It is critical to your stream security that no one other than ARServer can read the private file. As it's name implies, it must remain totally private to prevent a malicious person from creating a stream which appears to be coming from you. Once you have moved this file into a secure location, you will need to specify the path to the file in the RSP encoder settings PrivateKey field. If this field is blank, the no key is used and the stream is simple sent un-authenticated.

Finally, you will need a jSON advertising files that tell clients, relay servers and shoutcast bridges who to request a stream how to do so. An example of such a file can be found above in **The Protocol** section of this appendix. If you are using stream authentication, you will need to paste the public key that goes with the private key you created in the step above, into that field in the file. Again, the public key must be a single line with all the original new-lines replaced with "\n". Make sure the "Content" settings match the stream format and the "format" section matches the RSP settings you are using in the encoder (FEC Coding, Interleaving, etc). The IP4 section, (could be IP6 if you are using a IPv6 network) indicates how the connection should be accomplished. ReportHost fields, if included, tell where and how often receiver reports should be sent. If the Port field is zero, then it is expected that the connection will be made via a relay request through the relay server which is at the ReportHost address. If the Port is specified, then either the connection is through the specified Multicast group address, or if the Multicast field is missing or empty, then it is assumed that a static stream is already being being sent to the address of the listening machine on the specified Port number. Since ARServer RSP encoders are only for static streams, you must specify a Port property in a jSON file intended to direct a RSP listener to listen to the encoder, and that listener must be in the Addresses list.

If you are sending the stream from ARServer to a relay server or shoucast bridge, then you will need reference the location of this jSON file in the relay or bridge configuration settings so it knows how to receive the stream it is relaying or bridging, and the ARServer encoder Addresses list must include the address of the relay or shoutcast bridge so it actually sends packet to it. Again, the ARServer encoder does not relay streams, that the job of a relay server, so ARServer is sending only static streams to fixed addresses or to multicast group addresses. Your jSON file for will have to include the Port field.

You can create a jSON file and make it available on a web server for the general public to access. Such a file needs to direct listeners to connect to a relay server if you are not using multicast, so the jSON file will be a little different than the file used to listen to a static stream being sent directly from ARServer. The Port would be zero and the ReportHost properties would point to the relay server and relaying port number the relay server is listening for requests and reports on.

AudioRack Suite 3.4.1