Controlling Mirage Audio System

Controlling a Mirage Media Server

Introduction

We categorize control of a Mirage Media Server (MMS) into four categories:

- Connecting and the *preamble*
- Events
- Control
- Browsing & content selection
- Presets
- Playlists
- Now Playing
- Album Art

For information regarding using the RESTful endpoints, see the end of this document.

Connecting

An MMS is controlled via a socket or telnet connection to port 5004 on the device's IP address. Commands sent and responses received are terminated with a carriage return and a line feed. Commands commonly used in a connection preamble will be briefly defined here but will have a more complete definition later.

A typical initialization sequence looks like:

```
SetClientType DemoClient
SetClientVersion 1.0.0.0
SetHost 192.168.0.100
SetXmlMode Lists
SetEncoding 65001
SetInstance Player_A
SubscribeEvents
GetStatus
```

`SetClientType DemoClient` identifies the control client to the MMS. This command takes a single string argument.

`SetClientVersion 1.0.0.0` allows the control client to set a version. We *strongly* recommend setting a client version. This will allow the server to react to client version changes if necessary. This command takes a version string in the format **MAJOR.MINOR.BUILD.REVISION**

`SetHost 192.168.0.100` tells the server which address the control client connected on. This is useful if the control client is connecting through a external connection where the address it used might a web address. We recommend setting this value always. This command takes a single string argument

`SetXmlMode Lists` tells the MMS to send any lists as XML instead of text mode. This is recommended if the control client supports XML. This command takes a string argument where that argument is `None`, `Lists`, or `All`.

`SetEncoding 65001` tells the MMS to send data as UTF-8. There are other encodings available but `65001` will be the encoding of choice most of the time.

`SetInstance Player_A` sets which output subsequent browse and control commands are intended for.

`SubscribeEvents` tells the MMS to send events related to the currently selected instance as they occur. This command takes an optional boolean argument or a comma delimited list of events to limit the subscription to. If missing, the value is assumed to be `true`, which subscribes the control client to all events. The client will remain subscribed for the duration of its connection. The subscription will follow the client from instance to instance. No resubscription is necessary. Events for the selected or default instance are pushed to the connected client. To get events from another instance, see `SetInstance`.

`GetStatus` requests that all events related to the selected instance be sent now.

Events

The MMS communicates data back to its control clients through events. These events carry information about the current state of the server. They can also be used to request input from the user. Some events inform the control client about the availability of various functions while others tell the control client to take some action.

Events follow a simple format:

```
    Marker Source Event=Value
```

and will look like:

```
StateChanged Player_A TrackTime=121
```

or

```
ReportState Player_A TrackTime=121
```

Metadata Events

There are four lines of metadata and four metadata labels.

`MetaData1` is generally reserved for radio station names or for track count data.

`MetaData2` is generally reserved for the artist name.

`MetaData3` is generally reserved for the album name.

`MetaData4` is generally reserved for the track name.

`MetaLabelx` events will always provide the label for the corresponding `MetaDatax` field. `MetaLabel1` follows `MetaData1`, `MetaLabel2` follows `MetaData2`, etc. There are four `MetaLabelx` events

An example of these events:

```
ReportState Player_A MetaLabel1=
ReportState Player_A MetaData1=Pandora: Stevie Ray Vaughan Radio
ReportState Player_A MetaLabel2=Artist
ReportState Player_A MetaData2=Stevie Ray Vaughan
ReportState Player_A MetaLabel3=Album
ReportState Player_A MetaData3=Texas Flood (Legacy Edition)
ReportState Player_A MetaLabel4=Track
ReportState Player_A MetaData4=Texas Flood
```

Now Playing Art is handled by providing a few separate events.

`NowPlayingGuid` provides the ID of the now playing item. For example, `{20dd901a-b092-3386-dc16-6b56f38a811e}`

`BaseWebUrl` provides the protocol, address, and port portions of the URL to retreive art from. For example: `http://192.168.0.59:5005`.

For further details, see the Album Art section below.

Track Time

Track time is provided in seconds. A track duration *may* be provided, depending on the source of the content.

`TrackTime` indicates track position in seconds as a non-negative integer.

`TrackDuration` indicates the total number of seconds in the track as a non-negative integer.

In cases where the content does not have a total time (like a broadcast radio station from TuneIn), `TrackDuration` will be `0`.

In such cases where the MMS has neither a track length nor a current track position, both `TrackTime` and `TrackDuration` will be `0`.

Flags

All these events hold a `true` | `false` value and indicate whether a certian functionality is available.

`Back` defines whether or not there is anything in the navigation stack. If true, use `Back <int>` to jump back `<int>` number of pages. The navigation stack begins with `0`. `0` is the current page.

`BrowseNowPlayingAvailable` defines whether a queue is available to browse. This will be `true` when the queue has more than `0` items even the now playing item is a radio station.

`ContextMenu` defines whether or not `AckButton CONTEXT` is a valid command and the TuneBridge^{tm} button should be shown.

`Mute` defines whether or not the set instance is muted.

`PlayPauseAvailable` defines whether or not the `Play`, `Pause`, and `PlayPause` are valid and the Play or Pause button should be shown.

`RepeatAvailable` defines whether or not `Repeat` is a valid command and the Repeat button should be shown.

`Repeat` defines whether Repeat is enabled or disabled.

`SeekAvailable` defines whether or not `Seek` is a valid command and the scrubbing thumb should be shown on the track progress meter.

`ShuffleAvailable` defines whether or not `Shuffle` is a valid command and the Shuffle button should be shown.

`Shuffle` defines whether or not Shuffle is enabled or disabled.

`SkipNextAvailable` defines whether or not `SkipNext` is a valid command and the Skip Next button should be shown.

`SkipPrevAvailable` defines whether or not `SkipPrevious` is a valid command and the Skip Previous button should be shown.

Multistate Flags

Some values have more than two states and therefore cannot be represented as `true` | `false` values.

`ThumbsUp` indicates the state of the Thumbs Up button and whether the command is available. `ThumbsDown` is identical to `ThumbsUp`, replacing `Up` with `Down` in all cases. Possible states are `-1`, `0`, and `1` where:

`-1` indicates the button is disabled and the command is not available.

`0` indicates the button is enabled but not set and the command is available.

`1` indicates the button is both enabled and set and the command is available.

At the time of writing, no online service still uses a Stars rating system. However…

`Stars` indicates the state of the stars and whether the command is available. States range from `-1` to `5` where:

`-1` indicates stars are disabled and the command is not available.

`0` - `5` indicate stars are enabled and should be showing the number of stars indicated in the value. The command is also available.

Control

`Play` will tell the MMS to play.

`Pause` will tell the MMS to pause.

`PlayPause` will toggle the playstate between play and pause.

`Seek <int>` where `<int>` is a either a non-negative integer between `0` and the value of `TrackDuration` or a negative integer between `-1` and `-1 * <valueOfTrackDuration>`. When

`<int>` is non-negative, the playback position will be moved relative to the **start** of the track. When `<int>` is negative, the playback position will be moved relative to the **end** of the track.

`SkipNext` skips to the next track. This command is governed by the `SkipNextAvailable` event.

`SkipPrevious` skips to the previous track if the value of `TrackTime` is less than `5`. Otherwise, it restarts the current track if allowed by the given service. This command is governed by the `SkipPrevAvailable` event.

`ThumbsUp` toggles the `ThumbsUp` state between `0` and `1`. This command is governed by the `ThumbsUp` event.

`ThumbsDown` toggles the `ThumbsDown` state between `0` and `1`. This command is governed by the `ThumbsDown` event. On some services, setting the `ThumbsDown` state to `1` will also skip to the next track.

Browsing

Basics

All browsing on an MMS is done through the same basic format.

```
Browse<Container> <start> <count>
```

where `<Container>` is the target browse type, `<start>` is the one-based index the returned list should start from, and `<count>` is the number of items the returned list should contain at most. For example:

`BrowseArtists 1 10` will return `10` artists starting at item `1`.

`BrowseArtists 11 10` will return `10` artists starting at item `11`.

Depending on whether `SetXMLMode` is set, the response may be in either text mode or XML mode. As above, we strongly recommend XML if the control environment supports it as it offers more information.

Every browse item has a default action based on its type. We'll go over these default actions later on. These default actions can be superceded with attributes on each item.

`action` defines the secondary action to perform if the user presses the the action button for that item.

`listAction` defines the action to peform if the user presses the list item itself.

`browseAction` defines the action to perform **after** doing the default action for that item or doing the

`listAction` if one exists.

Other list item attributes

`artGuid` provides the guid to use if displaying art in the browse (See the Album Art section below). If this attribute is missing, use the value of the `guid` .

`button` provides an integer value that indicates which secondary action to offer on that item. The value of this attribute is defined by this table

Value	Function
0	Off
1	Add
2	Delete
3	Play
4	Power
5	PowerOn
6	Edit
7	AllTracks
8	ShuffleAll

`dna` provides the name of the attribute containing the value to use for display.

`guid` provides the item's ID for use in any action.

`hasChildren` indicates whether that menu item is a branch (`1`) or a leaf (`0`).

`name` provides the name of the item.

Picklists

Picklists are a way for the server to generically present a list to the control client without the control client needing any additional information about that menu. All online content uses Picklists. Some local menus are Picklists. Given their frequency, we must go over them first.

Picklists are always browsed using the `BrowsePicklist` command. In some cases, performing an item's

`action`, `listAction`, or `browseAction` might result in a Picklist when that item's default action would not. A good example of this is when selecting local content for playback. The list items each have a `listAction` that results in a intent clarification menu as a picklist that is sent to the client without the need of `BrowsePicklist`.

Picklist items are selected with the `AckPickItem <guid>` command, respecting the presence of the `listAction` attribute, of course.

Local Content

Local content can be browsed in whatever order is desired by the control client. However, most clients follow this pattern:

Albums => Tracks
Artists => Albums => Tracks
Composers => Tracks
Favorites
Genres => Albums => Tracks
Playlists => Tracks

To browse a top level list, simply clear the Music Filter using `SetMusicFilter Clear`, then send the appropriate Browse command. Valid commands are listed at the bottom of this section.

Online Content

All online sources are Picklist trees branching from `BrowseRadioSources`. The response to `BrowseRadioSources` is a list of `RadioSources`. To select a specific service, use `SetRadioFilterSource=<guidOfService>`. Subsequently, use the value of the intended online service. This example assumes `SetPickListCount` has been set. The example browses Pandora.

```
SetXmlMode Lists

BrowseRadioSources
<RadioSources total="9" start="1" more="false" art="false" alpha="false" displ
ayAs="List" caption="Radio sources" np="1">
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000002000" name="Deezer" dna
="name" isSearchable="True" browseAction="BrowseRadioGenres" button="0" />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000010000" name="iHeartRadio
" dna="name" isSearchable="True" browseAction="BrowseRadioGenres" button="0" /
```

```
>
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000008000" name="Murfie" dna
="name" isSearchable="True" browseAction="BrowseRadioGenres" button="0" />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000000010" name="Pandora Int
ernet Radio" dna="name" np="1" isSearchable="False" browseAction="BrowseRadioS
tations" button="0" />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000000008" name="SiriusXM In
ternet Radio" dna="name" isSearchable="True" browseAction="BrowseRadioGenres"
button="0" />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000001000" name="Slacker Rad
io" dna="name" isSearchable="True" browseAction="BrowseRadioGenres" button="0"
 />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000000100" name="Spotify" dn
a="name" isSearchable="True" browseAction="BrowseRadioGenres" button="0" />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000004000" name="TIDAL" dna=
"name" isSearchable="True" browseAction="BrowseRadioGenres" button="0" />
    <RadioSource guid="fbbcedb1-af64-4c3f-bfe5-000000000020" name="TuneIn Radi
o" dna="name" isSearchable="True" browseAction="BrowseRadioGenres" button="0"
/>
</RadioSources>

SetRadioFilter Source=fbbcedb1-af64-4c3f-bfe5-000000000010
BrowseRadioStations
<PickList total="23" start="1" more="true" art="true" alpha="false" displayAs=
"List" caption="Pandora Internet Radio" source="Pandora" sourceTop="1">
    <PickItem guid="e82779bd-058f-6478-4785-0bba198d3c1e" name="Account: demo@
example.com" dna="name" hasChildren="1" button="0" artGuid="fbbcedb1-af64-4c3f
-bfe5-000000000010" />
    <PickItem guid="ea03ab84-c1bc-3eb7-c6e8-5e2317c616cb" name="Create a Pando
ra station..." dna="name" hasChildren="1" button="0" singleInputBox="1" />
    <PickItem guid="126e079c-3c5b-8dad-017a-9bc61ca8e7db" name="Browse Genre S
tations" dna="name" hasChildren="1" button="0" />
    <PickItem guid="31323232-3637-3532-3237-373536373538" name="QuickMix" dna=
"name" hasChildren="0" button="0" artGuid="31323232-3637-3532-3237-37353637353
8" />
    <PickItem guid="30353433-3735-3932-3531-393737393330" name="Rage Against T
he Machine Radio" dna="name" hasChildren="0" button="6" artGuid="30353433-3735
-3932-3531-393737393330" action="action" />
```

```
    <PickItem guid="37373233-3237-3933-3437-333630353239" name="Stevie Ray Vaughan Radio" dna="name" hasChildren="0" button="6" artGuid="37373233-3237-3933-3437-333630353239" action="action" />
    <PickItem guid="34313932-3130-3333-3437-373937343738" name="Dinner Party Radio" dna="name" hasChildren="0" button="6" artGuid="34313932-3130-3333-3437-373937343738" action="action" />
    <PickItem guid="36303133-3235-3930-3135-393434383737" name="All Time Low Radio" dna="name" hasChildren="0" button="6" artGuid="36303133-3235-3930-3135-393434383737" action="action" />
    <PickItem guid="31373033-3236-3835-3935-383032373737" name="Sum 41 Radio" dna="name" hasChildren="0" button="6" artGuid="31373033-3236-3835-3935-383032373737" action="action" />
    <PickItem guid="32353233-3334-3132-3339-303834383336" name="Of Mice &amp;amp; Men Radio" dna="name" hasChildren="0" button="6" artGuid="32353233-3334-3132-3339-303834383336" action="action" />
</PickList>
```

Valid Browse Commands

`BrowseAlbums`

`BrowseArtists`

`BrowseComposers`

`BrowseFavorites`

`BrowseGenres`

`BrowseNowPlaying`

`BrowsePicklist`

`BrowsePlaylists`

`BrowseRadioSources`

`BrowseTitles`

`BrowseTopMenu`

Presets

To store a Preset, use the `StorePreset` command. This command takes an optional double-quoted name argument. If the name argument is specified, the MMS will store the Preset with that name. If no name is specified, the MMS will prompt for a name using an InputBox. InputBoxes are natively supported by all our control system drivers. As with all protocol commands, each command should be terminated with a carriage return and a line feed (\r\n).

Examples:

`StorePreset` - This will be responded to with an InputBox from the server.
`StorePreset "Party Time"` - This will save a Preset called Party Time

To recall a Preset, use the RecallPreset command. This command takes either the double-quoted name of the Preset or the unique ID of the Preset. To get either of these, please see the BrowseFavorites command, described below. Recalling a Preset will replace the state of the selected Instance with the state stored in the Preset. As with all protocol commands, each command should be terminated with a carriage return and a line feed (\r\n).

Examples:

`RecallPreset "Party Time"` - This recalls the Preset by name
`RecallPreset 9f9c8919-f939-d67a-dce2-cb049a4ead99` - This recalls the Preset by unique ID

Edit a preset with `EditPreset nameOrId`

Rename a preset with `RenamePreset nameOrId newName`

Delete a preset with `DeletePreset nameOrId`

To browse available Presets, use the `BrowseFavorites` or `BrowsePresets` commands. This Browse command adheres to the same pattern as all other Browse commands.

This feature was once called Snapshot.

Playlists

Playlists are browsed with `BrowsePlaylists`. This command adheres to the same pattern as all Browse commands.

Rename a playlist with `RenamePlaylist oldName newName`

Delete a playlist with `DeletePlaylist nameOrId`

Reorder tracks within a playlist with `ReorderPlaylist playlistId srceTrackId destTrackId`

Now Playing

The now playing queue can be browsed with `BrowseNowPlaying`. Only browse the queue if the `BrowseNowPlayingAvailable` event is `true`.

All now playing indexes are 1 based.

Change songs by using `JumpToNowPlayingitem <indexOfItem>`.

Reorder the queue with `ReorderNowPlaying <indexOfTrackToMove> <indexToMoveTo>`.

Remove a song with `RemoveNowPlayingItem <indexOfTrackToRemove>`.

Clear the queue with `ClearNowPlaying`.

Album Art

`BaseWebUrl` provides the protocol, address, and port portions of the URL to retreive art from. For example: `http://192.168.0.59:5005`.

The specific handler on the MMS is called `getart` so an example of a base for retreiving art would be `http://192.168.0.59:5005/getart?`. Always include the ID of the item.

To construct the full URL to get art, use the above values along with the below options.

Option	Purpose	Possible Values
c	constrain	0=size image to fit height and width 1=constrain to dimension and maintain aspect ratio
guid	unique id of the album, artist, genre, title, etc	
fmt	image format. Valid values are `png` or `jpg`	
instance	the instance GUID	
h	image height	

w	image width	
rfle	reflection elevation	
rflh	reflection height	
rflo	reflection opacity	
rz	reflection rotation (z axis)	

RESTful API

Introduction

The root endpoint for all RESTful communcation is `http://ipOrNameOfServer/api/`. A `GET` request to this endpoint will receive a JSON response containing three arrays: `events`, `browse`, and `messages`.

Response

`events` contains `name`, `value` pairs where the event names are the same as the previously described IP protocol and values carry the same meaning per event.

`browse` contains the response to any browse commands made on separate `GET` requests to this same endpoint. This will discussed more later.

`messages` contains generic messages.

```
{
    events: [
        {
            name: "TrackTime",
            value: 369
        }
    ],
    browse: null,
    messages: null
}
```

Sending commands

To send commands, simply replace any spaces in the command as it would be used in the IP protocal with a `/`. For example, if the IP command is `SubscribeEvents True`, a `GET` to `http://ipOrNameOfServer/api/SubscribeEvents/True` would execute that command. Be sure to URL encode any parameters to ensure no invalid characters

It's important to remember that there is a fundamental difference between the RESTful API and the IP protocol in that RESTful communications are **NOT** guaranteed to be processed in the order the control client sends them as they are all transmitted on different sockets. The server may receive them out of order. To handle this and to force sets of commands to be processed in order, make use of the `Script` command. This command allows several commands to executed in a specific order by transmitting them all at once. When using the `Script` command, be sure to URL encode each (sub)command entirely as it is a *parameter* rather than a command. For example, if we wanted to set the target instance and subscribe to events in that order, we would do `http://ipOrNameOfServer/api/Script/SetInstance%20Player_A/SubscribeEvents%20True`.

Regardless of the command sent, the response will be received in the next request to the bare endpoint `http://ipOrNameOfServer/api/`. It's important to note that because of the relationship between requests and their responses, only one RESTful session per computer is supported at this time.