SPINITRON AUTOMATION INTEGRATION

Document version: 2010-10-08

www.spinitron.com
617 233 3115
Contact: eva@spinitron.com

1 Introduction

This document describes integration of Spinitron with supported automation systems. Section 2 has a subsection for each system describing technical implementation, possible difficulties and implications that station management should consider. Section 3 discusses matters common across the systems. Section 4 is reference material.

FAIR WARNING: Remote logging from an automation system to Spinitron is a messy business. That's why this document is long—we describe all aspects of the mess that we are aware of and consider relevant so you can make informed decisions. A fraction of the material would suffice merely to configure an automation system so that remote logging "works" but proper disclosure requires much more.

With the instructions below, you can make your automation system send log messages to Spinitron. But Spinitron ignores them unless we configure it to accept them from your station. So contact technical support to get automation logging turned on. Moreover, we would prefer to review and test your configuration together with you before you use it routinely.

We wrote the text for a reader who has experience administering the automation system in question and has good general computer competence. There may be unfamiliar concepts here but we use only industry-standard jargon so you should have little trouble finding definitions and explanations online (Wikipedia is good on computer topics).

Computer strings and code are in this `monospace␣font` with dotted border and in which the space character is represented by a descending open box.

"We" refers to Spinitron the company/people while "Spinitron" refers to the Spinitron servers/software/system. "You" means you the reader or your station or someone at your station following the instructions.

2 Supported automation systems

We describe, in this section, how to set up remote logging to Spinitron on each of the supported automation systems. (For some of them there's more than one way to do it and each is described.) This will get you going quickly and shouldn't require much thought or background knowledge.

But following the "how-to" part in each section we have:

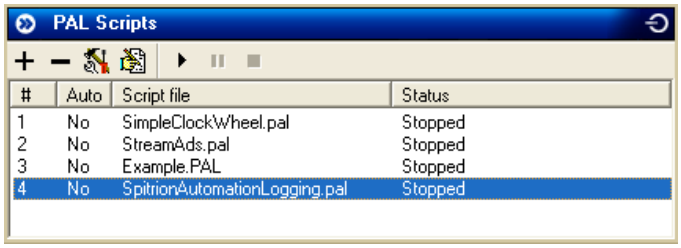
- Detailed discussion of problems with the system/method, if any
- Description of controlling Spinitron's playlist handling mode
- A summary of the automation system (and logging method) with respect to the general problems of remote automation logging

And you may need to read (parts of) the deeper discursive material in Section 3 to make sense of this stuff. This structure results form our decision to put the "how-to" for each system up front.

2.1 SAM Broadcaster (Spacial Audio Solutions, LLC)

SAM Broadcaster doesn't have any built in features specifically for sending song metadata to remote machines (unlike several other automation systems) but it does have PAL, a powerful built-in programming language with which we can implement what we need.

We wrote a basic PAL script that logs songs to Spinitron over HTTP (see Section 4.1). Feel free to adapt this to your situation and needs but, to maintain compatibility with Spinitron, please observe the comments indicating what not to modify.



Configure the script for your needs (by changing the constant values at the top) and save the file somewhere on the SAM computer with an obvious descriptive file name (because the file name is used as script name in the PAL Scripts window) and `.pal` extension, e.g.: `SpitrionAutomationLogging.pal`.

Then, in SAM, locate the PAL Scripts window (Desktop B in the default configuration of desktops) and click the + button in its toolbar. SAM asks you to locate the script file—do that and press OK.

To start/stop the script: click to select it and use the triangle/square buttons in the toolbar. The fourth icon from the left in the toolbar opens the built-in script editor and debugger.

You can configure the script to work in either full automation or live assist mode (see Section 3.2) with the `pm` constant. If you need to switch between modes on one computer you can, at the very least, have two PAL scripts, one that logs songs with `pm=0` and the other with `pm=2`. Be careful to only run one at a time. But, with more sophisticated PAL programming, you may be able to find a much nicer solution.

Table 1. *SAM Broadcaster summary—see Section 3 for exposition*

Label	Yes	Label names in the SAM library are logged to Spinitron
Playlist mode switch	OK	Switch between two scripts in the PAL Scripts window. Better methods possible with better PAL scripts
Protocol	HTTP	HTTP is our preferred protocol
Framing/delineation	Reliable	Nothing to worry about
Security	None	User/pass in plaintext in HTTP requests
Viability	Good	Solution is viable if station accepts security risks

2.2 Simian 2.0.7 (Broadcast Software International)

There are two ways to integrate Spinitron with Simian:

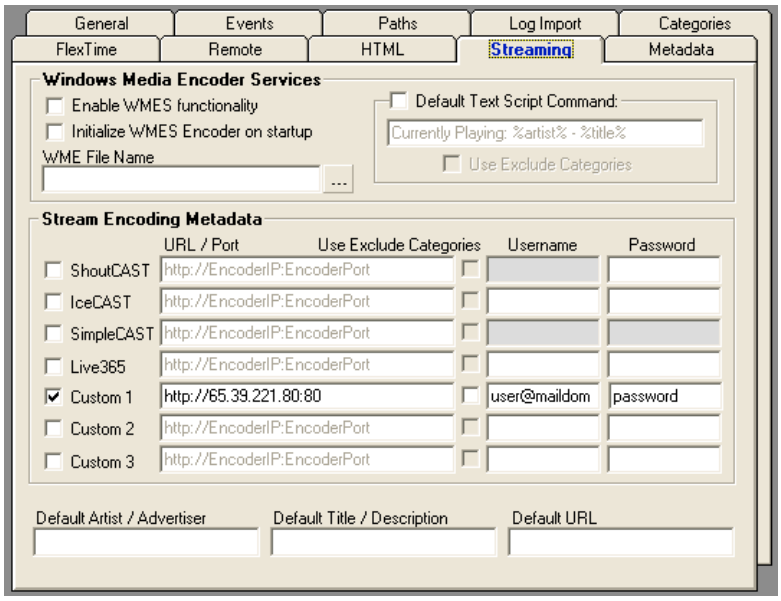
- ***HTTP method***: use Simian's Stream Encoding Metadata feature to send song metadata to Spinitron in HTTP requests
- ***PAD over TCP/UDP method***: configure it to send PAD Metadata messages to Spinitron over TCP or UDP

You configure these features in Program Options on the Streaming and Metadata tabs respectively.

2.2.1 HTTP method

The Stream Encoding Metadata feature is documented fairly well in the Simian manual. You configure it in two steps:

First, configure one of the three Custom services in the file `C:\BSI32\EncoderData.ini`. The configuration is a URL template string in which special tags such as `%ARTIST%` and `%TITLE%` are placeholders for song metadata. Every time a new song starts playing, Simian replaces the tags in the URL template with the corresponding song metadata and then sends an HTTP request for that URL to the server. There is a template for Spinitron logging in Appendix Section 4.3 in which you need to change the `st` parameter to your station ID.



Next, under Stream Encoding Metadata in the Streaming tab in Preferences, enable and configure the corresponding Custom service: ether 1, 2 or 3, depending what you did in `EncoderData.ini`.

In URL/Port, enter Spinitron's server IP address (resolve spinitron.com to find out, or ask our customer service) and port number 80 using the http URL scheme, e.g. `http://65.39.221.80:80`.

Enter the username and password of the Spinitron user account you will be using for automation playlist logging. (Your station ID is also required for authentication but it has to go in the URL template file).

We have not been able to see any effect from the Use Exclude Categories checkbox (possibly a bug). According to the manual, you ought to be able to use this together with Simian's Categories configurations to prevent anything but music from being sent to Spinitron. This would be a very good thing. If you figure it out, let us know.

You can operate in full automation mode with the URL template as given or append `&pm=2` for live assist mode (see Section 3.2). If you need to be able two switch between the two modes, you can configure two custom Stream Encoding Metadata services in `EncoderData.ini`, one with `pm=0` in the URL template and the other with `pm=2`. Then identically configure the two corresponding custom services in Streaming tab in Preferences. Enable one and disable the other with the checkboxes on the left to switch modes (never enable both). Simian playback must be stopped to access Preferences.

Problems in Simian 2.0.7

UPDATE—WE HAVE TESTED A BETA VERSION OF SIMIAN 2.1.0 THAT FIXES THE FOLLOWING BUGS. ONCE IT IS RELEASED, HTTP WILL BE THE PREFERRED METHOD.

There is a bug in Simian: it does not URL-encode the album name when constructing a URL with the `%album%` tag in the template. This is serious and we cannot use this feature for automation logging until Simian fixes it.

The Simian manual doesn't mention that you can use the `%publisher%` tag in the URL template but it works, albeit suffering from the same bug as `%album%`—it is not URL-encoded.

Another bug: if the HTTP server does not respond, Simian's user interface hangs up. Although the audio continues to play without interruption, it seems to the user as though Simian has frozen. It's hard to get out of this state without forcibly closing Simian. This might not be a showstopper but it's something you need to know.

So, as it stands, we can't use the HTTP method. That's unfortunate because it's a much better protocol (see Section 3.3) than PAD over either TCP or UDP.

Table 2. *Simian 2.0.7 HTTP method summary—see Section 3 for exposition*

Label	Bugs	Label names in Simian library are logged to Spinitron but with bugs
Playlist mode switch	OK	Switch between streaming metadata services in Preferences. Better UI with Macros and Custom Carts?
Protocol	HTTP	HTTP is our preferred protocol
Framing/delineation	Bugs	Album and label names not URL-encoded
Security	None	User/pass in plaintext in HTTP requests
Viability	None	Bugs prevent this method's use

2.2.2 PAD over TCP/UDP method

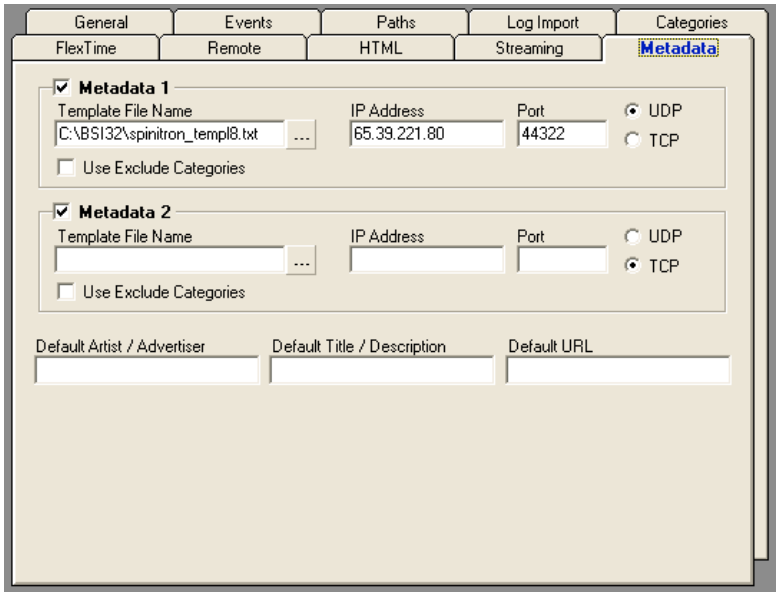
PAD is short for Program Associated Data, broadcast industry jargon for just what we need: metadata describing the program.

Simian's PAD Metadata feature employs a user-defined template file. The template file may contain arbitrary text and special tags that are placeholders for song metadata such as `%artist%` and `%title%`. (The Simian manual describes the tags as having the form `<!--BSIARTIST-->`, `<!--BSITITLE-->` etc. but these don't work.) Every time a new song starts playing, Simian reads the template file, replaces the tags with the corresponding song metadata to create a *PAD message*, and then sends it to a remote host over TCP or UDP.

Section 3.3 describes general problems with TCP or UDP transmission of PAD messages over the Internet. But in addition to these, Simian's current version implements PAD over TCP so that it stops when the TCP connection gets into certain (common enough) states.

So, with bugs preventing use of HTTP or PAD over TCP we are stuck with PAD over UDP, which is intrinsically unreliable. If the packet loss rate between Simian and Spinitron is low enough then it will work. But that's a huge "if" clause and we have no way to predict how much of the time it will be true. For now, it's all we've got.

To set up PAD over UDP, save the Spinitron PAD template file (Appendix Section 4.4 or get the latest from us) on the Simian computer, e.g. in `C:\BSI\`. Then configure the PAD Metadata feature in he Metadata tab of Program Options as follows.



Enable either service 1 or 2 with the checkbox. Click the … button and select the Spinitron PAD template file you saved in the previous step. Enter Spinitron's server IP address (resolve spinitron.com to find out, or ask our customer service) and port number 55431. Select UDP with the radio button.

The Use Exclude Categories checkbox is as described for the HTTP method above (i.e. baffling).

To use live assist mode (see Section 3.2) you need to add a line with `^pm=2` to the template. To switch between full automation and live assist on one computer you need to change the template file or switch between two template files. If both Metadata 1 and 2 (in Program Options: Metadata tab) are available then you can configure 1 with a template including `pm=0` and the 2 with `pm=2`. Enable Metadata 1 and disable 2 (with the enable checkbox's in Program Options: Metadata tab) for full automation and visa versa for live assist (and never check both at once!). There may be a way to use Simian's macros to make the process easier for DJs.

Table 3. *Simian 2.0.7 PAD over TCP/UDP method summary—see Section 3 for exposition*

Label	Yes	Label names in the Simian library are logged to Spinitron
Playlist mode switch	OK	Switch between PAD metadata services in Preferences. Better UI with Macros and Custom Carts?
Protocol	TCP/UDP	TCP is unusable owing to session bugs. UDP is unreliable
Framing/delineation	Good	Not 100% reliable but probably good enough
Security	None	User/pass in plaintext in PAD messages
Viability	Fair	Solution is viable if UDP message loss rate is low in practice and station accepts security risks

2.3 DAD (ENCO Systems, Inc.)

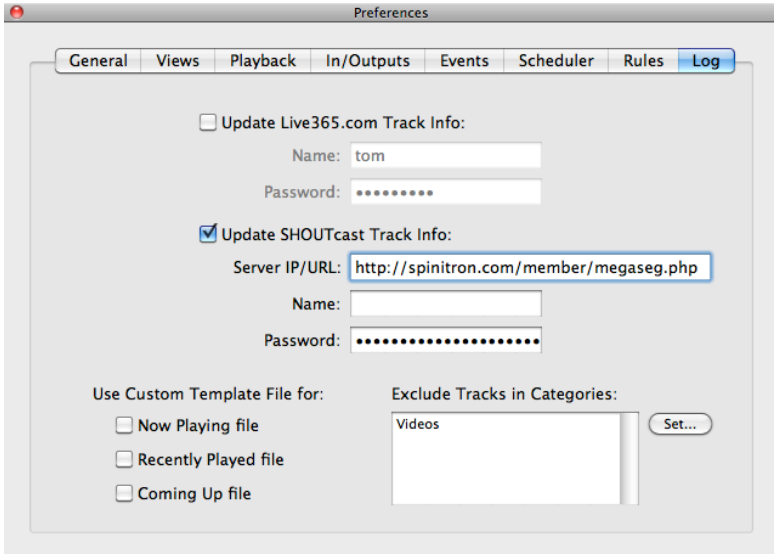
2.4 Megaseg 5.1 (Fidelity Media, Inc.)

UPDATE—MEGASEG 5.5 HAS GENERALIZED LOGGING OF NOW PLAYING METADATA OVER HTTP. IT REPLACES THE SHOUTCAST METHOD AND IS SUITABLE FOR OUR PURPOSE. THE TEXT BELOW NEEDS TO BE UPDATED ACCORDINGLY.

There are two ways to integrate Spinitron with Megaseg:

- ***Shoutcast method***: redirect Megaseg's Shoutcast metadata update messages to Spinitron
- ***Now Playing file method***: Run a program that monitors Megaseg's Now Playing file and sends the song metadata it contains to Spinitron

Megaseg's Shoutcast and Now Playing file features are both configured in the Log tab in Preferences.



2.4.1 Shoutcast method

You can type an arbitrary URL into the Shoutcast "Server IP/URL" field and Megaseg will, every time a new song starts, request (with the HTTP GET method) the URL from the server with metadata of the now-playing song in the GET query parameters. Assuming the station isn't already using this Shoutcast feature, it can be used to log songs to Spinitron.

The "Name" field is (as far as we can tell) ignored. The value in the "Password" field is included in plaintext as one of the query parameters. A typical query part of the request URL looks like this:

```
?mode=updinfo&pass=password&song=I%27m%20Insane%20-%20Sonic%20Youth%20-
%20Bad%20Moon%20Rising%20%281985%29
```

There are three GET parameters in the query: `mode` is a Shoutcast command and irrelevant to our purpose, `pass` is obvious, and `song` contains all the (URL-encoded) song metadata, which decoded is:

```
I'm␣Insane␣-␣Sonic␣Youth␣-␣Bad␣Moon␣Rising␣(1985)
```

Notice how Megaseg has strung Title, Artist and Album (from the Megaseg library) together with two space-hyphen-space `␣-␣` separator substrings.

This poses two problems. First, entirely absent is record label (or Publisher as it is denoted the Megaseg library), which is relevant to SX reporting. It's not necessarily a killing point but stations that care about the label field should think carefully about what happens when it is omitted from the automation log messages sent to Spinitron. Section 3.1 deals with this matter.

Second, in typical music metadata, space-hyphen-space is a common enough substring to cause trouble delineating the artist, song and disk fields in the `song` parameter. We don't know an algorithm that can *reliably* do this. Again, this might

not be a killing point but some stations will surely find it irksome or worse. To get a measure of the problem, take a look at a large iTunes music library, type a hyphen into the search field and scroll through looking for space-hyphen-space substrings.

Authentication: to log the song in a playlist, Spinitron needs to identify the station that sent the log message and authenticate a Spinitron user account to own the playlist. Type station, username and password all into Megaseg's Shoutcast Password field separated by the pipe symbol, e.g.

```
wxyz|user@maildomain.com|password
```

You might find it easier to compose this in another program and paste into Megaseg because Megaseg obscures typing in the password field.

To control playlist handling mode on the server, i.e. full automation vs. live assist modes (see Section 3.2), we can use configuration either of the Shoutcast "Server IP/URL" field or of another parameter coded into the password field. These are the only places available to put the information that allows the client to select the mode. This might be acceptable for static configurations. But if one instance of Megaseg on one computer at the station does double duty then switching between the modes will likely be uncomfortable for DJs. Reconfiguring the hidden password is just not realistic so we are stuck with reconfiguring the URL, e.g. switching between `megaseg-auto.php` and `megaseg-live.php`.

Table 4. *Megaseg Shoutcast method summary—see Section 3 for exposition*

Label	No	Label names in the Megaseg library are **not** logged to Spinitron
Playlist mode switch	Poor	Reconfigure Shoutcast Server IP/URL in Preferences
Protocol	HTTP	HTTP is our preferred protocol
Framing/delineation	Poor	Space-hyphen-space in artist or song name breaks delineation
Security	None	User/pass in plaintext in HTTP requests
Viability	Maybe	Solution is viable if (1) logging label name is not required, (2) occasional artist/song/disk name delineation failures are acceptable, and (2) station accepts security risks

2.4.2 Use the Now Playing file

If you click the checkbox for "Now Playing file" under "Use a custom template for:" (in Megaseg's log preferences), Megaseg prompts for a template file. The template file may contain arbitrary text and special tags that are placeholders for song metadata. The tags are special HTML comments such as `<!--MegaSeg␣Artist-->` or `<!--MegaSeg␣Title-->`. Every time a new song starts playing, Megaseg reads the template file, replaces the tags with the corresponding song metadata and writes the Now Playing file into the Megaseg Logs directory.

Spinitron wrote a script that runs in the background monitoring the Now Playing file. Every time Megaseg changes the Now Playing file, the Script detects the update, reads the file, extracts the song metadata and sends it to Spinitron in an HTTP or HTTPS request.

This overcomes the two problems with the Shoutcast method (missing label and poor field delineation). First, we have designed a Now Playing file template (see Appendix section 4.2) that should provide satisfactory field delineation. Second, there is a `<!--MegaSeg␣Publisher-->` tag for label name. (See also Section 3.1).

You need to modify the template file using your station ID and user/pass for your automation logger user account.

A concern with our script is its lack (at present) of a conventional graphical user interface: it runs on the Mac OS X command line (CLI), which may or may not be a problem. If, on your station's automation computer, you use Megaseg *only* in full automation mode (see Section 3.2) then we can set up the script to run as a system daemon running all the time and all songs played by Megaseg on that computer will be logged on Spinitron in playlists belonging to the automation user account. But if you need to switch between full automation and live assist modes on one computer then we need to come up with some kind of user interface.

Table 5. *Megaseg Now Playing file method summary—see Section 3 for exposition*

Label	Yes	Label names in the Megaseg library are logged to Spinitron
Playlist mode switch	Poor	Requires use of CLI. GUI development a possibility?
Protocol	HTTP(S)	HTTP(S) is our preferred protocol
Framing/delineation	Good	Not 100% reliable but probably good enough
Security	Good	HTTPS can be used
Viability	Depends	Solution is viable if playlist mode switching is not required. Otherwise viability may depend on software development

2.5 Rivendell Open Source Radio Automation

The fine folk at WMFO (Somerville, Mass.) wrote a loadable extension module for Rivendell 1.5 called Spinitron Update. They released[1] it as open source under GNU GPL 2.0 so you are free to use it, modify it and contribute to its further open source development. It has been in use for some time so we have confidence in its robustness.

You need to edit the source code to enter your station ID and the user/pass of the automation logger account and recompile the module. We have no experience with

[1] https://wiki.wmfo.org/Operations/Code/Rivendell_-_Spinitron_Update

the recompile part but, if you are successfully using Rivendell already, you probably do.

The module supports both live assist and full automation modes of operation. If you use the same computer for both of these roles at your station (as is the case at WMFO) then you will need to train staff to switch modes as appropriate. WMFO's wiki web page[1] says that they implemented the user interface in Rivendell using macro carts.

If, on any given machine at your station, Rivendell runs exclusively in either live assist or full automation mode then you have a couple of options. You can use the module as is and set configure Rivendell into the mode you need or you can modify the source and recompile the module hard coded into one mode or the other, loading the appropriate module(s) in your instance(s) of Rivendell.

Table 6. *Rivendell summary—see Section 3 for exposition*

Label	Yes	Label names in the Rivendell library can be logged to Spinitron
Playlist mode switch	Good	Custom macro cart
Protocol	HTTPS	HTTP(S) is our preferred protocol
Framing/delineation	Reliable	Nothing to worry about
Security	Good	HTTPS is used
Viability	Proven	Field proven

3 Topics common to all automation systems

3.1 Logging label name (aka publisher)

If logging label name is important to you, perhaps because you want to use the automation-logged playlists in SoundExchange reports, then two problems need to be addressed:

- Does the automation system you use allow logging of label name?
- Do you have any (or enough) label names in your automation music database?

If the answer to either question is no then you will end up sending song log messages to Spinitron without label name. We discussed the first question for each of the automation systems (and transmission method) in turn in Section 2 but the technical capacity to send label names is useless without label names in the automation music database.

The situation raises more questions:

- Can you get label names into your automation system's library?
- What happens when you log to Spinitron without label name?
- Can Spinitron fill in the blank label names?

It appears DAD does not deal with label name or publisher but all the others (SAM, Simian, Megaseg and Rivendell) support a suitable publisher/label field and have ways to populate that field. So (DAD aside) the answer to the first question is, in principle, yes.

How you would populate it is another matter. Entering it manually would be relatively reliable and accurate but you may not have the resources. Gracenote CDDB could accurately identify label name when it rips a CD but it usually does not because it doesn't have the data. That situation is compounded and made stable because iTunes doesn't have anywhere in its library to store label name.

Other commercial music metadata databases are available (e.g. Amazon) that include label names. Could you refer to one of these to look up the label name? It's possible but the results would be badly inaccurate. First, there will be mismatches between your automation library's entries and corresponding ones in the database in which you're searching for label names. Second, many songs have been released several times on different labels, often in different mixes and versions but with the same or similar name. If the point of logging label name in Spinitron is SoundExchange reports then accuracy is relevant because it affects who gets the royalties for use of the sound recording.

Spinitron performs a lookup a bit like this in the manual playlist entry web form. The autocomplete feature may fill in the label name form field if the user selects both artist and disk name suggestions. But in this case, the human user is still responsible for the data entered. If the suggested label name is wrong, it should not be used.

But if Spinitron were to implement something similar, to fill in absent label names in automation log messages there would be no human oversight and any number of errors could creep in. Such a feature is certainly possible but we would need to extract disclaimers form clients who want to use it—Spinitron could take no responsibility for the accuracy of the results.

3.1.1 Spinitron's processing of a log messages without label name

Under very specific conditions, Spinitron will assign a label name to a song logged by an automation system when there's no label name in the message. We set out the conditions in detail here and discuss how it might be useful.

Database structure

The database schema used by each Spinitron member station has (among others) the following tables:

- Artists
- Disks
- Labels
- Songs

A system of references among the rows in these (and other) tables constructs the playlists and other views of your data that you are familiar with as a Spinitron user.

Rows in Artists are joined to rows in Disks by a fifth table called Appearances. One row in Appearances represents one *appearance* of one artist on one disk. This structure allows a many-to-many relationship between disks and artists: one artist can *appear* on many disks and one disk can have *appearances* of many artists.

Each row in Songs references an *appearance*, which allows a many-to-one relation between songs and the appearance of an artist on disk.

And finally, each row in the Disks table can reference a row in the Labels table allowing a many-to-one relation between disks and a label.

Logging songs to the database

When an automation system logs a song to Spinitron, the message it sends includes an artist name ***A***, a disk name ***D***, and a song name ***S***. We are considering the case that the message does not include a label name. Spinitron uses the same algorithm to add

the song to your database that it uses when a DJ enters a song manually by the pressing Submit button in the playlist entry form.

What happens depends on whether or not there is a *matching appearance* of ***A*** on ***D*** already in the station's database. To be precise:

- If there already exists in the database an *appearance* of ***A*** on ***D*** then *there is a matching appearance.*
- If ***A*** isn't already in Artists then *there is no matching appearance.*
- If there are no rows in Disks matching ***D*** then *there is no matching appearance.*
- If ***A*** exists in Artists and there's one or more rows in Disks match ***D*** in Disks but none of them already have an appearance of ***A*** then *there is still no matching appearance*, except in a special case…
    - …if the log message indicates that ***D*** is a multi-artist compilation an existing row in Disks matches ***D*** that is also a multi-artist compilation, then Spinitron *adds a matching appearance* (of ***A*** on ***D***) to the Appearances table.

Those are the two ways to get a *matching appearance*. In the three other cases, a *new appearance* is created. Finally ***S*** is entered into the Songs table and assigned to the *matching* or *new appearance*. So the song ***S*** will appear in its playlist[2] with a label name if ***S*** gets assigned to a *matching appearance* of an artist on a disk that's already assigned to a label.

Consequences

Stations that need label for compliance reasons might wonder if they can exploit the behavior described above to fill in label if the automation system can't send it or doesn't have it. It can be done but most likely only with special preparation.

Remember, to fill in the label:

- You need a *matching appearance*,
- already existing in your station's database,
- in which the disk must already have a reference to a label.
- And the search for artist and disk names requires an exact match[3]

This is a lot different from autocomplete and web search. The only way we can imagine to reliably get a good hit rate is to import your music library *with* label names

[2] And therefore also in SoundExchange and various other reports and views.
[3] We mean exact in the sense that no fuzzy matching is used, not to imply that strict string comparison is. In fact some flexibility is involved: matching is case insensitive and it ignores diacriticals. See: http://www.collation-charts.org/mysql60/mysql604.utf8_general_ci.european.html

into Spinitron in advance and use the same music library as the basis for the automation system.

That's only going to be helpful under certain circumstances:

- Your music library is small enough that you can manually add label names to it before importing to Spinitron
- You have a lot of volunteers to share that work
- You're willing to constrain the automation system to play from the subset of songs in the library with label names
- You use automated fuzzy matching against a commercial database to find label names (mentioned in previous section) and accept the associated inaccuracies
- You library has label names but your automation system won't import them or can't log them to Spinitron

For example, to support SoundExchange quarterly reporting you might: 1. Use volunteers to add label names to a starter set of disks in the existing library. 2. Keep that project rolling over time. 3. Enforce a policy that new music added to the library must have label name. 4. Configure the automation system to play only songs with label during your SX reporting periods.

3.2 Playlist mode: full automation or live assist?

Automation systems can be used in these two modes:

- ***Full automation***: playing the station's program content with no DJ required, or
- ***Live assist***: making it a manually operated media playback device that a DJ uses while doing a radio program live on air.

Spinitron's remote logging capability supports both of these by offering the automation system control over how Spinitron selects (or creates) the playlist into which any given song is logged.

You may have noticed that in Section 2, all methods of transmitting song metadata to Spinitron involve organizing them into strings identified by codes, most of them with two letters: `st`, `sn`, `dn` etc. These are then sent either as HTTP GET parameters, or as form data in HTTP POST requests, or as labeled strings in a formatted text file.

An additional playlist handling mode parameter `pm` provides the required control over Spinitron's playlist selection. The three modes are, in simple terms:

`pm`	Used for	Playlist handling
`0`	Full automation (this is the default if pm	Log the song to a suitable playlist belonging to user identified in he message, creating a playlist

	is omitted or invalid)	if necessary
1	Full automation	As `pm=0` but also mark the playlist as the "current playlist"
2	Live Assist	Log the song to the "current playlist"

You can ignore mode 1. Since the first release of the remote logging API, Spinitron's use of the "current playlist" concept has diminished to near irrelevance. (If you care, the change can be found in Appendix section 4.7.1.) You can also probably ignore the formal specification of the effects of `pm` on server behavior in Appendix section 4.5.

If `pm=0` then Spinitron logs the song into a playlist belonging to the automation logger user account (i.e. the one specified together with the song metadata in each log message your automation system sends to Spinitron).

If `pm=2` then Spinitron inserts the song the "current playlist". When a DJ opens a playlist choosing "*I am working on the playlist:* ***live on-air***" then this playlist becomes the "current playlist". In this way the DJ can live-log a show in Spinitron and save the trouble of typing the song into Spinitron's web form whenever he or she spins a tune from the automation system in live assist mode. The user credentials un and pw are still required in the automation log message to authenticate the sender of the message even though the song is logged into a different user's playlist.

If on a given computer, the automation system is used exclusively in full automation mode then you can leave `pm` out of the configuration altogether. If the computer is used exclusively in live assist mode then `pm=2` can be hard coded into the URL or PAD template or PAL script.

But if a given computer at your station does double duty, performing full automation and live assist, *and* you want it to log songs to Spinitron in both modes then your staff needs a way to switch `pm` between `0` and `2`. And you really want to avoid using the automation system if `pm` is set wrong. There are notes on controlling mode in each subsection of Section 2.

3.3 Protocols between automation systems and Spinitron

There are at present four available protocols for the transmission of remote logging messages from automation systems to Spinitron:

- HTTP
- HTTPS (i.e. HTTP over secure TLS/SSL transport)
- PAD over TCP
- PAD over UDP

Conveniently, HTTP and HTTPS are the same except for their use of security at the transport layer. Here we treat them as equivalent—the differences are discussed in Section 3.4.

3.3.1 HTTP

We consider HTTP to be much superior for our purpose (remote logging) than PAD over TCP or UDP because HTTP defines a robust session protocol.

In HTTP's case, the session protocol runs on top of a TCP (or secure TLS/SSL) connection. Generally and very roughly speaking, a session protocol (such as HTTP's) formally specifies how the entities involved in the communication should behave on matters such as: which entity initiates the connection, which entity talks first over the connection, how protocol compatibility between the two is verified, how transactions between the entities correspond to messages sent over the connection, what each entity should do in the sending and receiving of messages to reliably implement the transactions, how each entity handles errors or misunderstandings, how unresponsiveness or unexpected disconnection of the peer is handled, etc.

The session protocol also formally specifies the service it provides to the application, which, in HTTP's case, involves a client requesting a resource identified by a URL and a server responding with the resource.

None of these matters is trivial—designing a robust session protocol is hard!—and all Internet communication protocols need to address the problems.

HTTP is well designed, mature and robust and quality implementations are available as commercial or open source software. It is also well suited to our purpose: an automation system (the HTTP client) encodes song metadata as an HTML form and sent with either HTTP's GET or POST method[4]; Spinitron (the HTTP server) responds either with a confirmation or message identifying what went wrong.

3.3.2 PAD over TCP/UDP

PAD over TCP/UDP as implemented by many automation systems (including Simian and DAD) do not define a session layer. That's because they were designed to run not over the Internet but over a LAN, fire-walled from the Internet, in which negligible packet loss and latency can be taken for granted.

TCP

In any use of TCP, some kind of session protocol is unavoidable, whether it is explicitly defined or merely emergent in the software design. As far as we know so far, in every PAD over TCP implementation the session behavior is undefined. And

[4] While both work, we prefer the POST method on the grounds of proper HTTP compliance. According to Section 9.1 of the HTTP/1.1 specification (RFC 2616), we should not use GET because our use is, in HTTP terminology, not "safe". http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1

there appears to be no industry standard. But if you study an implementation (reverse engineering), a session protocol emerges from its various behaviors. For example, in any given implementation, answers must exist to the following (and many more) behavioral questions:

- Which entity (automation system or PAD receiver) initiates the connection?
- Is there one connection per PAD message or is a persistent connection used for many?
- How does each entity handle an unresponsive peer?
- Is reconnection attempted if the connection fails? Which entity is responsible? What are the procedures?
- If a persistent connection is used, how are messages delineated? Which entities are allowed to terminate the connection? Is there any idle timeout?
- etc.

A good session protocol may be easier to design for a safe, low-loss LAN than for the Internet but it is still not trivial. For example, in the case of Simian, we know that its session behavior causes problems. The manual says: "Note that TCP can cause delay/timeout issues if the address and port specified are not accessible". We have observed every-day conditions (e.g. the PAD receiver reboots) cause complete failure of Simian 2.0.7 PAD communications.

In other words, Simian's (emergent) session protocol is defective and not suitable for remote logging.

We have not yet tested DAD.

UDP

In PAD over UPD, there is little mystery about the session protocol, it is a null layer: the PAD message (and nothing else) goes in the UDP data payload and the datagram is sent and summarily forgotten about by the sender. Relative to TCP this is nicely unambiguous and predictable but there are two problems.

First, for our purposes, UDP (indirectly) limits the length of the PAD message. Routers are not obliged to fragment IP packets exceeding the next-hop link's MTU. So we have to assume that if the PAD message (plus UDP overhead) is too big to fit in an IP packet of length equal to the path MTU, the IP packet will not be fragmented and the message will not be delivered. PMTU discovery wouldn't help, even if the automation system implemented it, because there's no mechanism to adapt the PAD to fit the message length limit. Automation systems don't even have a way to use a statically configured limit. All we can do is try to keep the messages as short as possible and hope for the best.

In practical terms, this means that we should use PAD message templates with the minimum of overhead and eliminate non-essential fields. For example, where possible we would avoid XML. But we need to take care not to trim so much from the template that we compromise field delineation (which is a topic of the next subsection).

The second problem with UDP is much bigger—it is unreliable by design. If the message is lost in the network, neither the sender nor the receiver will know unless a session protocol provides for reliability, which is not the case with automation systems.

Message loss is a serious concern. In our application, PAD message loss will correspond closely to IP packet loss. About the worst thing about IP packet loss in Internet applications is its unpredictability. Most of the time, the loss rate is very low but sometimes it is significant and occasionally it is very high. There are no guarantees. The best we can do if we use UDP is audit the automation system log against the corresponding Spinitron playlists to identify the losses.

Message framing and field delineation in PAD messages

When we use, HTTP the solutions for message framing and field delineation are trivial. HTTP provides the transactions to accomplish message framing and music metadata elements are encoded as individual strings using the application/x-www-form-urlencoded type, which handles field delineation.

With PAD over TCP/UDP, on the other hand, we need to design a solution.

First, message framing: In PAD over TCP, if persistent connections are used then we need a pattern to identify the message within the TCP byte stream. XML is often used for PAD messages and it provides for framing assuming one PAD message exactly corresponds to one XML document.

XML is OK for TCP but we don't recommend XML for UDP owing to the limited UDP message length and XML's heavy overhead. Message framing in UDP is easy because we assume the UDP payload contains exactly one PAD message and nothing else.

Field delineation is also handled by XML but there remains a small concern about transparency of the field values. We must assume that music metadata string can be anything. So, in an XML template for PAD messages we would wrap CDATA elements around the automation system's tags.

But we further assume that the automation systems do not encode, escape or quote the strings with which they replace the tags (how could they?). In other words, we assume that metadata strings could break the field delineation and/or message framing.

In the case of XML, this means we have to accept that if the three character string `]]>` appears in any music metadata, the XML parser will fail and Spinitron will have to discard the message.

For PAD over UDP in Simian and the Now Playing file in Megaseg, we designed the file/message format shown in the corresponding appendices. We believe they will do no worse than XML at framing and delineation but with lower overhead.

3.4 Security

There are three security aspects to consider with remote logging over the Internet:

- Server authentication
- Client authentication
- Privacy of sensitive data

Server auth means that the client has a way to authenticate the server, i.e. to check that the server it is communicating with is indeed Spinitron. This is the same problem as being sure that, when you log on to a web site that purports to be PayPal, it really is PayPal and not some imposter. For automation remote logging, this may not be of great concern to you but Spinitron offers server auth when the remote logging is performed over HTTPS.

We achieve client auth by including a user name and password along with the song metadata in each HTTP request/PAD message from the automation system. This leads directly into the third security aspect.

Usernames and passwords are sensitive data and we normally expect them to be kept private. But these credentials travel over the network in plaintext in all available protocols (see Section 3.3) except HTTPS. It would not be hard for a motivated attacker to read the credentials from log messages without detection. With them, he or she has access to Spinitron as one of your station members with whatever permissions that particular account has.

HTTPS is a ubiquitous, trusted protocol that provides all three security features we need: server and client auth and data privacy. HTTP and PAD over TCP or UDP provide none of them.

At present only Rivendell and the Megasg Now Playing method use HTTPS.

Spinitron will accept log messages purporting to be from your station only from IP addresses or address ranges that you tell us to authorize. For example, if your only automation computer has a static public IP address then we can configure Spinitron to only accept messages for your station from that IP address. This makes injection attacks via the remote logging API a little harder but doesn't prevent an attacker from stealing account credentials with which to log on to the web interface.

Use of HTTP or PAD over TCP/UDP without additional security measures is inconsistent with all of Spinitron's other security procedures and precautions. It is up to your station's management to decide if the risks are acceptable. Relevant considerations include: What could motivate the attacker? Is the prize worth the effort? What harm could an attack do you, your station, staff, parent institution etc.?

4 Appendices

4.1 SAM logger PAL script

The following script is very basic. Feel free to embellish it if needed.

```
// Basic PAL Script to log songs to Spinitron when a new song starts
// playing in the active deck.

// Configuration. Change the values here according to your situation.
const station  = 'wxyz';                // your station
const username = 'user@maildomain.com'; // logger account username
const password = 'password';            // logger account password
const urlbase  = 'http://spinitron.com/member/samlog.php';
const minlen   = 30;        // don't log songs < minlen seconds

// Declare variables
var song: TSongInfo;        // instantiate a TSongInfo object
var url: String = '';       // in which the full URL is composed
var result: String = '';    // result string from Spinitron
var length: Integer = 0;    // song length in seconds

// Logger loop
while (True) do
begin
   { NOTE: From SAM documentation...
     ActivePlayer:
      Returns the player object that is currently playing a track.
      If DeckA is playing, it returns the DeckA player.
      If DeckB is playing, it returns the DeckB player.
      If both DeckA and DeckB are playing, then it returns DeckA.
      If neither DeckA or DeckB are playing the result will be nil.
   }
   if ActivePlayer <> nil then
   begin
      song := ActivePlayer.GetSongInfo; // DO NOT MODIFY song
      if song <> nil then
      begin
         length := Round(StrToInt(song['duration'])/1000);
         if length >= minlen then
         begin
            // DO NOT MODIFY FROM HERE DOWN TO...
            url := urlbase;
            url := url + '?st=' + URLEncode(station);
            url := url + '&un=' + URLEncode(username);
            url := url + '&pw=' + URLEncode(password);
            url := url + '&aw=' + URLEncode(song['artist']);
            url := url + '&sn=' + URLEncode(song['title']);
            url := url + '&dn=' + URLEncode(song['album']);
            url := url + '&dr=' + URLEncode(song['year']);
            url := url + '&dl=' + URLEncode(song['genre']);
            url := url + '&ln=' + URLEncode(song['label']);
            url := url + '&sc=' + URLEncode(song['composer']);
```



```
            url := url + '&se=' + URLEncode(song['comments']);
            url := url + '&sd=' + IntToStr(length);
            result := WebToStr(url);
            // ...HERE
            writeLn(result); // rudimentary logging
         end;
      end;
   end;
   PAL.WaitForPlayCount(1);
end;
```

4.2 Megaseg Now Playing template file

Use the following template file but check with us if it has since been updated.

```
~~~~MEGASEG1
~st=wzbc
~un=user@maildomain.com
~pw=password
~sn=<!--MegaSeg␣Title-->
~aw=<!--MegaSeg␣Artist-->
~dn=<!--MegaSeg␣Album-->
~Categories=<!--MegaSeg␣Categories-->
~cn=<!--MegaSeg␣Composer-->
~se=<!--MegaSeg␣Notes-->
~ln=<!--MegaSeg␣Publisher-->
~dr=<!--MegaSeg␣Year-->
~sd=<!--MegaSeg␣Length-->
~^^^MEGASEG1
```

4.3 Simian HTTP Stream Encoding Metadata URL template

There should be about 10 lines in the `C:\BSI32\EncoderData.ini` file. Change one of the custom lines to the following (changing `Custom1` at the beginning to `Custom2` or `Custom3` if needed).

```
Custom1=/member/simianlog.php?st=wxyz&un=%username%&pw=%password%&aw=%arti
st%&sn=%title%&ct=%category%&sd=%lengthseconds%&dn=%album%&ln=%publisher%
```

Change the value of the parameter `st` to your station ID. Append `&pm=2` to set Spinitron's playlist handling mode to live assist.

4.4 Simian PAD template file

Use the following template file but check with us if it has since been updated.

```
~~~~SIMIAN1
~st=demo
~un=tom@spinitron.com
```

```
~pw=himhowl
~category=%CATEGORY%
~aw=%ARTIST%
~sn=%TITLE%
~dn=%ALBUM%
~sd=%LENGTH%
~dl=%GENRE%
~dr=%YEAR%
~sc=%COMPOSER%
~ln=%PUBLISHER%
~curtime=%CURTIME%
~curdate=%CURDATE%
~se=%COMMENTS%
~^^^SIMIAN1
```

4.5 Spinitron generic remote logging API parameters

The table below details the complete set of *generic* parameters for remote logging.

Key	Name	Required (Y means yes. * means either `al` or `aw` must be specified)	
		↓	Notes
`st`	station ID	Y	ASCII string, typically a 4-letter station call sign
`un`	user name	Y	ASCII string, email address username of valid user account
`pw`	password	Y	ASCII string, password corresponding to user name
`pm`	playlist handling mode		Digit: 0, 1 or 2, default=0, explained in Playlist handling section below
`aw`	artist whole name	*	UTF-8 string, ignored if `al` is given, whole name of artist
`al`	artist last name	*	UTF-8 string required if `aw` is not given, alphabetical part of artist name
`af`	artist first name		UTF-8 string, part of artist name before alphabetical part
`sn`	song name	Y	UTF-8 string
`sc`	song composer		UTF-8 string
`se`	song note		UTF-8 string, arbitrary metadata attached to song
`sp`	song timestamp		any parsable ASCII time or date-time string, defaults to local time of station
`sd`	song duration		ASCII string, either the duration in seconds or in minutes and seconds separated by a colon, e.g. `4:33`
`rq`	request		if set to 1 flags the song as a request
`ne`	new		if set to 1 flags the song as new material at the station
`lo`	local		if set to 1 flags the disk as local music
`dn`	disk name		UTF-8 string, name of disk, defaults to song name
`df`	disk (media) format		UTF-8 string, e.g. "CD", "LP", "7-inch", preferably supply only values configured for the station's menus, defaults to unset
`dt`	disk (release) type		UTF-8 string, e.g. "Album", "Single", "Comp", preferably supply only

		values configured for the station's menus, defaults to unset
`dl`	library/genre	UTF-8 string, e.g. "Rock", "Jazz", "Blues", preferably supply only values configured for the station's menus, defaults to unset
`dr`	disk release year	4-digit year, year the disk was released, defaults to unset
`da`	disk add date	any parseable ASCII date string, date the disk was added to the station's libraries, defaults to unset
`ln`	label name	UTF-8 string, default to empty string
`lc`	label country	UTF-8 string, default to empty string
`le`	label email	email address, default to empty string
`lu`	label URI	web site address URI, default to empty string

We expect automation systems will usually send artist name in `aw` form rather than `af` + `al`.

All *generic parameters* (i.e. those in the table) are available to all automation systems. But Spinitron extends the *generic parameters* for *specific* automation systems with *automation system-specific parameters* where necessary. SAM and Rivendell use the generic API while Simian and Megaseg need some (automation system-) *specific* parameters. You can identify them in the appropriate subsection in Section 2 and corresponding appendices, for example in Simian there is `category`, `curdate` and `curtime`.

4.6 Character encoding

In the generic API, character encoding is either US-ASCII or UTF-8 depending on the parameter. Both of these are compatible with Windows Latin-1 (Windows-1252) and the ISO-8859 family of character encodings so long as only the ASCII range of characters is used and the upper half of the code page from 0x80 to 0xFF are strictly avoided, otherwise Spinitron's behavior is unspecified. For example, sending metadata names with extended Latin characters (such as é or ü) in one of these single-byte encodings will not work properly.

In practice, song metadata strings quite frequently use characters outside the ASCII range.

So, for each supported automation system, Spinitron has an API that performs encoding conversion *to the best of our current understanding of the automation systems*. The caveat is needed because the automation systems and their documentation are not explicit about their use of character sets and encodings.

4.7 Playlist mode parameter formal specification

When the Spinitron server accepts a song for logging with `pm=0` or `1` or with `pm` absent from the query, it shall select a playlist according to the following rules:

1 If a playlist exists, owned by user `un` and the song timestamp (i.e. `sp` or the local time at the station, if `sp` is not given) lies between this playlist's date and on-air time and off-air time, the server shall log the song to this playlist.

2 If more than one playlist matches the criteria in 1., the server shall select from these the playlist with the greatest playlist-ID (which is normally the most recently created).

3 If no playlist matches the criteria in 1., the server shall create a playlist belonging to user `un` with date corresponding to the song's timestamp for a show selected or created according to the following rules.

4 If a regular (i.e. scheduled) show owned by user `un` and the song timestamp is on one of the show's weekdays and between its on-air time and off-air time, the server shall create a playlist for this show and log the song to it.

5 If more than one shows matches the criteria in 4., the server shall select from these the one with the greatest show-ID (which is normally the most recently created).

6 If no show matches the criteria in 4., the server shall create an irregular show (i.e. not in the station's program schedule) owned by user `un`. The show's on-air time shall be the song timestamp rounded down to the nearest whole hour, its off-air time one hour after the on-air time and its show name shall be the Station ID in upper case followed by on-air and off-air times, e.g. "WXYZ 3am - 4am".

If the request includes `pm=1` and if the song is successfully logged then the server shall mark the playlist selected by the above rules as the station's current playlist (defined below).

Hence if a station uses computer automated operations for certain scheduled hours of the week, the automation user account may be given regular shows covering those hours and Spinitron will create playlists as needed and log songs into them according to song timestamp (or local time at the station). If an automation system logs songs outside the weekdays/hours of these scheduled shows then irregular shows with generic names are created.

4.7.1 Current playlist and pending current playlist

Since the first release of this API, struck-out text below has become obsolete.

The "current playlist" is the playlist that appears on the station's public Spinitron web site when the web client selects no playlist, e.g.

```
http://spinitron.com/radio/playlist.php?station=wxyz
```

~~shows wxyz's current playlist.~~

Spinitron now selects (for public display when the user has chosen nothing else) the playlist containing the song with the most recent date/time that is in the past.

Spinitron marks a playlist as the current playlist when:

1 a DJ logs a song into a playlist in the "live on-air" manual logging mode,

2 the station's local time matches the date and timestamp of a song in a playlist logged using "before the show" logging mode,

3 when a song is logged using this API and the query includes `pm=1`, or

4 when the station's "pending current playlist" variable is set and a song is logged using this API and the query includes `pm=2`. At the same time, the "pending current playlist" variable is cleared.

When a user opens a playlist in "live on-air" logging mode, the server shall mark this playlist as the station's "pending current playlist" ("pending" in the sense that there are no songs logged in it yet). When a song is logged into the pending current playlist, either manually using the web form or using this remote API with `pm=2`, then the server shall mark the playlist as the station's current playlist and clear the station's "pending current playlist" variable.

So it is important that the automation system not log songs with `pm=0` or `1` when a show is being performed manually and the DJ is logging songs manually. Songs played manually by a DJ on the automation system should be logged with `pm=2`.

