
This paper describes the design, implementation, and testing of an information system built to generate streaming radio reports for ibiblio and the open-source Icecast streaming media server. This new system allows end-users to generate reports on demand, in either standard Icecast format or the SoundExchange 2013 reporting format. The system was designed to help aid community radio stations in complying with government reporting and regulations. In addition, a plan to increase system availability for ibiblio radio streaming is discussed.

Headings:

- Distributed Systems
- Information Systems
- Radio Webcasting
- Cloud Services
- Web Databases
END-USER SYSTEM FOR GENERATING SOUNDEXCHANGE REPORTS

by
David A Cowhig

A Master’s paper submitted to the faculty of the School of Information and Library Science of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in Information Science.

Chapel Hill, North Carolina
April 2016

Approved by

_______________________________________
Paul Jones
Table of Contents

1. Introduction .................................................................................................................. 2
2. Background ................................................................................................................... 4
   2.1 Brief History of Radio ............................................................................................... 4
   2.2 Hardware Infrastructure ............................................................................................ 7
3. Literature Review ......................................................................................................... 10
   3.1 Government Regulations ......................................................................................... 10
   3.2 NPR/SoundExchange .............................................................................................. 12
   3.3 Icecast ..................................................................................................................... 13
   3.4 Information System Design .................................................................................... 14
   3.5 Rapid Application Development ............................................................................. 16
4. Use Cases and User Profiles ....................................................................................... 18
   4.1 Use Cases ............................................................................................................... 18
   4.2 User Profiles ........................................................................................................... 18
5. System Design ............................................................................................................. 19
   5.1 Original System ...................................................................................................... 19
   5.2 New System ............................................................................................................ 20
6. Implementation ............................................................................................................ 23
7. Testing and Results ...................................................................................................... 25
8. Limitations ................................................................................................................... 27
9. Future Direction ........................................................................................................... 28
10. Appendix A ............................................................................................................... 31
11. Appendix B ............................................................................................................... 35
12. Appendix C ............................................................................................................... 36
13. Appendix D ............................................................................................................... 37
1. Introduction

A webcast allows digital media, such as music or video, to be broadcast over the Internet. Like any traditional radio broadcast, a webcast can be picked up by many simultaneous listeners. However, distance limitations of wireless signals no longer dictates how far a broadcast can be transmitted. It also makes the initial entry into broadcasting easier for smaller stations and radio amateurs. Unfortunately, current government rules and regulations in the United States places a technical burden on these smaller webcasters if they want to play recorded performances to the public.

In compliance with the Digital Millennium Copyright Act (DMCA), webcasters are expected to report songs streamed on the Internet to a non-profit organization, SoundExchange. SoundExchange then pays out royalties to artists and copyright owners. Large radio streaming sites, such as iHeartMedia and Pandora, are able to hire a team of technical support staff to generate these reports. However, smaller stations do not always have a dedicated technical support staff member. Some local stations, such as community radio, are even operated by volunteers.

Up until recently, when a streaming radio station hosted by ibiblio needed access to their streaming logs, they had to submit a help request through the ibiblio website. That request would then go in a queue for a staff member to accept. This request could take up to 72 hours before it was acted upon, especially if the request was made on the weekend or over a university holiday. Once a staff member accepted the support request,
they would manually generate a report using their preferred Linux shell. Although a
Python script was ultimately created to assist ibiblio staff when responding to streaming
radio report requests, there was a lack of consistency among the different staff members.

A review of ibiblio support tickets was conducted to understand how these radio
reports were being generated. A sample of 30 support tickets showed that staff differed in
how they handled requests. For example, some staff members generated reports in their
home directories, while others generated them in the location of the Python script used to
create the report. Some staff members deleted the report once they were sent to the radio
station, however, other staff members would leave the reports on disk. This made it
difficult to determine what reports have already been run.

The method of delivering completed reports to stations also differed among staff
members. Some were compressed with the Unix tar command, while others were
compressed inside a zip file. This led to follow-up support requests by stations asking
how to open a file they were not familiar with. In addition to the compression differences,
some reports were emailed to the station, while others were sent using cloud storage, such
as ibiblio.org or Dropbox.

To resolve these issues, this paper describes an in-depth examination of the ibiblio
radio streaming hosting infrastructure. This examination includes an analysis of the
current rules and regulations required to report listening statistics to SoundExchange. It
also identifies approaches currently available to allow the automation of these reports. A
new system is then detailed, implemented, and tested that allows ibiblio to offer a single
streamlined reporting solution to its hosted webcasters.
2. Background

2.1 Brief History of Radio

Wireless telegraphy, better known as radio, got its start as a method for providing point-to-point communication. One of the most well-known pioneers of the type of communication was Italian inventor Guglielmo Marconi. In the late 19th century, Marconi created the Wireless Telegraph and Signal Company to manufacture and sell radio systems. An example of Marconi equipment in use at the time was the radio operated aboard the RMS Titanic in 1912. Although the Marconi radio installed on the Titanic was said to be one of the most advanced at the time, it had serious limitations (“Wireless Could Have,” 2012).

Point-to-point communication is similar to the modern-day telephone. One node initiates a call to another node. For example, a person dials a number and another person on the other end answers the request. This behavior may have saved more lives on the Titanic if it were available at the time. However, point-to-point communication in 1912 acted more like a “party line” (“Wireless Could Have,” 2012). In the case of the Titanic, multiple radio stations and ships were sending messages at the same time. This disruption of the radio signals is known today as jamming (FTC, 2012).

Unfortunately, this meant that messages from the Titanic became distorted or were lost before the intended receiver could get the message. Some messages were only partially received. In addition to the confusion of having multiple messages sent over the same signal, radio operators also did not have a complete set of standards when sending
messages over the radio. This was a fatal flaw in radio at the time, as certain pieces of
data, such as the Titanic’s latitude and longitude, were not correctly received (“Wireless
Could Have,” 2012).

In response to the Titanic tragedy, the United States enacted the Radio Act of
1912. This new law mandated that special frequencies be set aside for emergency
communications. It also created a “right-of-way” for these emergency communications.
All nodes, or point-to-point stations, were required to give priority to signals from ships
in distress. Although the Radio Act of 1912 would promote safer communication
standards for sea vessels, it also had other consequences at home in the United States.

While Marconi was working on point-to-point radio communication, amateur
hobbyists in the United States were testing another usage for radio. These amateur
operators were developing what is known today as broadcasting. Broadcasting, also
known as one-to many, is the model current-day radio stations use. In this model, a single
transmitter transmits a signal to multiple receivers and listeners. The first known
successful broadcast over radio was transmitted in 1906 (Slotten, 2006).

By 1912, the few radio broadcasts available consisted of music played on a
phonograph. Although broadcasting was popular among amateur radio operators,
commercial operators were still heavily invested in point-to-point communication radio
only (Slotten, 2006). The Radio Act of 1912 greatly limited amateur radio operators. The
new law required all radio operators to have a license. It also required that “private or
commercial station not engaged in the transaction of bona fide commercial business by
radio communication” to limit broadcasts to within a distance of 200 meters.
It would not be until 1920, when the station KDKA started broadcasting, that other commercial organizations saw the benefits of radio broadcasting for entertainment. Although radio broadcasting was dominated by corporations in the 1920s, it was researchers at universities that published most of the research on promoting the idea of radio broadcasting for public entertainment. Over 100 stations licensed at the time were owned by these same universities. Not surprisingly, university radio was mostly dedicated to public service broadcasting (Slotten, 2006).

University radio stations would go on to promote future innovations in radio. In 1994, WXYC, a university radio station, became the first to transmit their radio broadcast over the Internet (WXYC's Simulcast, 2004). This broadcast was made possible by hosting from ibiblio, a digital library and archive project run by the School of Information and Library Science and the School of Journalism and Mass Communication at the University of North Carolina at Chapel Hill. WXYC’s Internet broadcast came only one year after the first Internet radio broadcast by Carl Malamud in 1993 (Shoffner, 1994).

ibiblio is also the hosting provider to a few community radio stations. These stations provide their listeners with “engaging, local, relevant, and unique content” (Mustain, 2013). Unlike most commercial radio, these stations provide content relevant to their local communities. With Internet radio, these stations can broadcast beyond the distance limitation of low powered FM radio. Listeners who are traveling or have moved can still stay involved in their local community from anywhere with an Internet connection.
2.2 Hardware Infrastructure

The ibiblio streaming infrastructure hardware is monitored around-the-clock from a secure data center with redundant power sources and environmental controls. The data center offers both 1Gbps and 10Gbps Internet connections. Current statistics provided by the ibiblio system shows this hardware has supported over 1,800 listeners at peak times for 2016. As of the writing of this paper, ibiblio has 19 individual streams connected from radio stations. Below is a simplified diagram of hardware infrastructure for Icecast:

![Diagram of hardware infrastructure for Icecast]

*Figure 1: Example Web Radio Streaming Hardware Design*
The first item on Figure 1 (A) is the connected listeners. Radio stations that broadcast over the internet need a way for listeners to receive the signal. Thanks to digital media format standards, many devices connected to the Internet can receive the signal. The most common example is a standard desktop personal computer (PC). Using software, such as Apple iTunes, Microsoft Windows Media Player, or VideoLan Client (VLC), listeners can quickly listen in on a public webcast.

Listeners also have the convenience of listening to a webcast on mobile devices. This includes: laptops, tablets, and smartphones. With a wireless data subscription, listeners can connect to a webcast away from a wired Internet connection. With the spread of low-cost hardware, and the creation of new easy-to-use open-source software, such as Google’s Android OS, Internet media receivers are finding their way into many electronic devices.

However, in order to receive a webcast, there needs to be a webcaster. Shown on Figure 1 (B), radio stations have equipment that will take a broadcast and send it to the ibiblio streaming server. This equipment, known as an encoder, can simply be software on a computer, or a dedicated piece of hardware. Either way, the encoder will take an audio signal as an input and stream it to the Icecast server. The Icecast server software is then responsible for making the broadcast available to listeners.

Figure 1 (C) shows a network switch. Although this piece of equipment may seem trivial to most users, it has a critical task to successful radio webcasting. Every network connection, whether from a listener or radio station encoder, must pass through the network switch from the Internet. A low-cost or defective switch could bring a webcast to a halt, especially when under high load. Switches with a high blocking factor are also a
In terms of performance cost, bandwidth from the Internet is one of the most important factors in operating a web radio streaming system. Cost and Internet speed will vary depending on physical location and provider (Mack & Rayburn, 2005).

Icecast is the streaming server software ibiblio utilizes to provide webcasters with a method for broadcasting a radio station over the Internet. At the time this paper was written, the current version of Icecast, which is distributed under the GNU Project’s General Public License (GPL), version 2, is 2.4.3. Figure 1 (D) is an example of where Icecast would fit in a web radio streaming infrastructure. Icecast hardware requirements are minimal. A 3GHz dual-core server with 2GB of random access memory has been shown to easily handle 14,000 listeners (Icecast, 2005).

Finally, as seen on Figure 1 (E), a radio streaming infrastructure needs some type of persistent data storage. The ibiblio infrastructure utilizes a network storage device that communicates over the Network File System (NFS) protocol. NFS allows multiple hosts to access a single storage pool. Although Icecast does not require a large amount of storage to operate, ample persistent storage is required to store and archive Icecast logs. These logs will be the source of data for the radio reports generated and submitted to SoundExchange.
3. Literature Review

3.1 Government Regulations

Starting with the 1912 Radio Act, the United States government not only mandated radios to be installed in ships carrying 60 or more people, but they also required radio operators to have a license. This set the groundwork for the government to make future rules and regulations (Slotten, 2006) that would impose fines and penalties to stations that failed to comply with them. An example of such a change was the equal-time rule in the 1927 Radio Act, which allowed political opponents to request the same amount of on-air time that their rivals received. This 1927 law also formed the Federal Radio Commission (FRC) to oversee the radio broadcasting industry. In the years following the 1927 Radio Act, the FRC was dissolved and became part of the Federal Communications Commission, also known as the FCC (Howeth, 2016).

To this day, the FCC is still tasked with regulating radio broadcasting in the United States. Over the years, the United States government and the FCC have been faced with challenges when it comes to regulating new technology. A well-known example is digital music. In 1999, the term “MP3” was the number one term queried on Internet search engines (Rose, 2002). MP3 is a digital audio coding file format. It allows listeners to store their music collection on digital devices, including the PC (Frankel, Greely, & Sawyer, 1999). With the rise of peer-to-peer (P2P) file sharing, such as Napster, MP3 files were easily shared among many users of the P2P network. It did not take long until
the Recording Industry Association of America (RIAA) started to take legal action (Borland, 2003).

The concern from the RIAA was music piracy. Music piracy is the redistribution of an original recording without permission. The lack of permission constitutes a form of copyright infringement, which is a violation of the United States copyright law (see Appendix B for copyright duration of musical compositions). In 2001, it was reported that record sales dropped by 10% “with an estimated 3.6 billion songs being downloaded via the Internet every month” (Rose, 2002). Although it has been argued that P2P file sharing does not harm the music industry (Peitz & Waelbroeck, 2004), the RIAA continued to aggressively pursue legal action.

The RIAA has a few legal statutes it can call upon when seeking to remedy a suspected case of music piracy. In 1998, the United States Congress instituted a number of new laws and regulations, including the Digital Millennium Copyright Act (DMCA) and the Copyright Term Extension Act (CTEA). The DMCA created procedures to follow in the event of suspected piracy on the Internet. As described in these procedures, a notification of violation is sent to hosts of suspected pirated content. This notification is known as a “takedown” request. As a result of the DMCA, many requests to have media removed from sites on the Internet have been made. The DMCA has been used to remove both illegal and legal content from the Internet (McIntosh, 2013).

Also included in the DMCA are provisions to regulate “eligible non-subscription services” webcasters. These webcasters, which mostly rely on advertising and donations to operate (Hart, 2012), are now qualified to obtain a compulsory license. As part of the
new license availability, stations are required to report broadcast information to SoundExchange, which was part of the RIAA (Rose, 2002) until 2003.

3.2 NPR/SoundExchange

SoundExchange is not only tasked with “administer[ing] the statutory license, which allows services to stream artistic content while paying a fixed rate for each play,” (“General FAQ”, n.d.) but they are also in charge of collecting royalties. These royalties are then distributed between artists and copyright holders. As part of an agreement with the Corporation for Public Broadcasting (CPB), NPR Digital Services (NPR/DS) serves as a “single collection point” between public radio stations and SoundExchange. This agreement also creates a flat rate for royalties.

According to NPR/DS, SoundExchange requires all stations to report two weeks of streaming logs and playlists. The playlists are created and managed by the station, and are therefore straightforward to report. The streaming logs, however, are not. This is because different streaming servers have different logging formats. In response to the various file formats generated from different streaming servers, SoundExchange created a standard format for the 2013 reporting year (Kempf & Theriault, 2012).

The new 2013 SoundExchange format requires streaming logs to be reported in a tab delimited text file with the following headers:

1. IP address (#.#.#.#; Do NOT include port numbers (127.0.0.1:3600))
2. Date listener tuned in (YYYY-MM-DD)
3. Time listener tuned in (HH:MM:SS; 24-hour military time; UTC time zone)
4. Stream ID (No spaces)
5. Duration of listening (Seconds)
6. HTTP Status Code
7. Referrer/Client Player

Figure 2: Sound Exchange 2013 Headers
Failure to report in this new 2013 format will result in a fee of $500.00 per reporting quarter. In addition, if a station has multiple streams, they must report in a single text file. NPR/DS claims that the standard format helps them negotiate favorable terms with SoundExchange (Kempf & Theriault, 2012). A standard format also allows stations and copyright holders to build systems that will be interoperable in the future. The ability to share this information between systems could save time and money in the event CPB was defunded (“General FAQ”, n.d.) or if SoundExchange were to lose their status as the authoritative reporting organization.

3.3 Icecast

As mentioned in section 2.2, ibiblio uses the open-source Icecast streaming media server to stream webcasts. Icecast is a small, yet powerful, program for reliable media streaming. As it is an open-source project, Icecast is available for download in source or binary form at icecast.org. It comes prepackaged for many popular Linux distributions and the Windows operating system. Anyone with an Internet connection is free to download and install it at no cost.

Icecast operates over the standard Hypertext Transfer Protocol (HTTP) protocol. This means many modern web browsers with media support, such as Google Chrome or Mozilla Firefox, can listen to a stream without any additional software installed. Icecast is also capable of displaying system status over HTTP in a web browser. For example, if enabled, a user can see streams available on the server, what is currently playing, and how many users are connected.

Finally, Icecast is capable of logging user connections and activity on the server. According to the documentation on the Icecast website, three log files are available for
storing data. These log files are named: access, error, and playlist. The access log simply logs all requests to the server from a client in the Apache Common Log Format. Below is an example (Grace, Maheswari, & Nagamalai, 2011) of the Apache format:

<table>
<thead>
<tr>
<th>Host</th>
<th>Ident</th>
<th>User</th>
<th>Date/Time</th>
<th>Request</th>
<th>Status</th>
<th>Referrer</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.123</td>
<td>-</td>
<td>-</td>
<td>[15/Jan/2016]...</td>
<td>GET wbib</td>
<td>200</td>
<td>Mozilla</td>
<td>143</td>
</tr>
<tr>
<td>192.168.1.99</td>
<td>-</td>
<td>-</td>
<td>[15/Jan/2016]...</td>
<td>GET wbib</td>
<td>200</td>
<td>Apple</td>
<td>266</td>
</tr>
<tr>
<td>192.168.1.101</td>
<td>-</td>
<td>-</td>
<td>[15/Jan/2016]...</td>
<td>GET wbib</td>
<td>200</td>
<td>SunOS</td>
<td>108</td>
</tr>
</tbody>
</table>

*Table 1: Apache/Icecast Raw Format*

The Icecast access log can vary in size depending on the number of connections a stream gets in one day. Smaller radio stations may have only a few hundred lines in a log file, while a mid-sized station may have in the low millions. Although the access log contains all the data needed to generate a SoundExchange report, it is not in the correct format. Each line of the log file will need to be parsed separately. Once parsed, the line can be rewritten to the SoundExchange 2013 format (see Figure 2), as shown below:

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Date</th>
<th>Time</th>
<th>Stream</th>
<th>Duration</th>
<th>Status</th>
<th>Referrer</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.123</td>
<td>2016-01-15</td>
<td>10:16:11</td>
<td>wbib</td>
<td>143</td>
<td>200</td>
<td>Mozilla</td>
</tr>
<tr>
<td>192.168.1.101</td>
<td>2016-01-15</td>
<td>17:01:33</td>
<td>wbib</td>
<td>108</td>
<td>200</td>
<td>SunOS</td>
</tr>
</tbody>
</table>

*Table 2: SoundExchange 2013 Format*

### 3.4 Information System Design

Langer (2007) writes that a reason many information system projects fail is due to either 1) a lack of planning, or 2) unrealistic planning. The author goes on to add that project managers and developers can avoid this common mistake with proper planning methodology. One well known methodology is the Software Development Life Cycle, or SDLC. When using the SDLC, developers have steps to follow to complete the development of a new system. The generic steps are as follows:
1. Determine the need for a system to assist a business process
2. Define that system’s goals
3. Gather business requirements
4. Convert business requirements to system requirements
5. Design the database and accompanying applications
6. Build, test, and implement the database and applications

Although many software developers like to start writing code as the first step in any project, the SDLC calls for sitting down with the client. This includes evaluating any existing systems and performing interviews with those who will be using the system. Doing so helps determine the requirements and needs of the client. Ideally, we want to build a system that improves on a current business process. By taking the time upfront to understand the current workflow, use cases, and setting realistic goals, we are greatly improving our chances of building a successful system. It’s also much easier to make changes early on before the system is in use.

Once the requirements and goals are selected, feasibility of the new system needs to be determined. Based on the information gathered by setting requirements and client needs, it may become apparent that some software already exists to perform the function the client is seeking. It may also be determined that building a new system is just too costly. If the project is determined to be feasible, it can then move on to the design phase, knowing that due diligence has been completed to help prevent risk and reduce the possibility of system failure.

In the design phase, Graphical User Interface (GUI) screens, inputs, and outputs are modeled. Models of databases and workflows are also drawn. Designers want to focus on usability and accessibility to maximize system use. Finally, this information will be written as a detailed report of specifications for the new information system. Only after these specifications are completed can the programmers start coding.
The typical SDLC methodology works well for larger organizations with a team working on the new system. However, what if we are developing a simple system for a smaller organization or individual? A method for speeding up the development process is completed by combining the analysis and design phases. This methodology is known as Rapid Application Development, or RAD (Langer, 2007).

### 3.5 Rapid Application Development

Beynon-Davies, Carne, Mackay, & Tudhope (1999) describes the software design methodology known as RAD. RAD, as its name implies, attempts to create software applications in a short amount of time. It does this by using an iterative design which relies heavily on user input throughout the design process (Beynon-Davies & Holmes, 2002). Working with users, programmers focus on small pieces of the whole system. Part of what helps RAD cut down on development time is that programmers are paired with users who are knowledgeable about the needs of the software system being designed.

Brooks (1987) explains how software development methodology needs to take into account changeability. Unlike traditional product design and manufacturing, software can change after it has been published. Software may require an update to correct glitches found in the code. Updates to user requirements will also alter the design or features of the software. A positive side effect of RAD is increased changeability. For example, knowledge gained during design and testing can be immediately be inserted into the software. The need to wait until a software package is released is no longer necessary in order to get valuable user feedback.
However, building information systems using RAD isn’t always straightforward (Agarwal, Prasad, Tanniru, & Lynch, 2000). Beynon-Davies & Holmes (2002) use the issue of breakdowns in RAD to understand how to improve on the process. They employ Heideggerian terminology to explain how a breakdown occurs in an information system: A breakdown occurs when the system changes “from being ready-at-hand to being present-at-hand.”

Ready-at-hand refers to a system that is functioning properly. When a user interacts with it, it performs tasks as expected. To the user, the system becomes transparent, and its inner-workings go unnoticed. However, when the system fails, it becomes present-at-hand. Present-at-hand refers to a system that is in the way, or blocking the user from completing their desired task. The operation of the system, or lack thereof, becomes visible to the user.

To overcome breakdowns found, Davies & Holmes (2002) argue that “an application is always complete, but never finished.” The idea of taking breakdowns into account early on in the development process goes back to the point made in Books (1987). The never-ending changeability of software and the “accidental tasks” that appear in new system development cannot be avoided. Not only does this require the SDLC to be iterative, but it also extends this cyclical design style to RAD.
4. Use Cases and User Profiles

4.1 Use Cases

There are currently two use cases for the new system:

1. A radio station that needs to submit streaming radio reports to SoundExchange every quarter. As this is the most common case, it will be the default option when accessing the system.

2. A station that needs access to raw icecast log files. Some stations may want to keep internal copies of streaming metadata. They may also need it to report to organizations other than SoundExchange.

4.2 User Profiles

Users of the new system will have various computer hardware and operating systems in use within their organization. In addition, the level of computer knowledge will vary from user to user. For these reasons, the new system specifications call for a web-based application that will run on many platforms. This will accommodate the various computer platforms being used. To aid in the adoption of the new system, the process of generating reports should be similar to manually requesting them from ibiblio staff. The process is as follows:

1. Identify the station name
2. Select two-week interval for SoundExchange, or daily for raw data
3. Receive a file with requested data
5. System Design

Using RAD methodology, three radio station staff members (clients) were selected to help aid in the new system development and testing. The first two clients have requested radio streaming reports in the past from staff through the ibiblio online help form. The third client was new to SoundExchange reporting, and had no interaction with the workflow of the previous system. Initially, the third client requested “two concurrent weeks of typical programming streaming logs” to submit to SoundExchange.

5.1 Original System

The original system relied on a custom script written in Python 2.6 on CentOS 5. The script, labeled as version 1.0, took in three arguments on the command line. Argument 1 was the start date. Argument 2 was the stop date. Argument 3 was the station name. Once started, the script would open log files from the NFS share which matched the date range from argument 1 through 2. The script would then scan each line in the logs and look for a match. A match was determined by the station name supplied in argument 3.

If a match was found, the script would perform some basic checks to make sure the data found on the line was valid. If the line did not contain all the data needed, it would be skipped, otherwise, the data was inserted into an array in memory. This array was then written out with a new format to a new file. This process would continue until the report was complete with all dates requested in the command line arguments.
Unfortunately, version 1.0 of the script was not well documented. It was designed to report in the SoundExchange 2013 only. There was also another copy of the script that generated reports in the default icecast Apache format. Based on discussions found in the ibiblio ticket system archive, it appears that a new student staff member did not notice there were two scripts, the first for SoundExchange format and the second written to meet the needs of radio stations that wanted the file in raw icecast format. This led to a new version of the script, 1.1, written in Python 2.7 on CentOS 6.

The new version of the script added to the number of arguments it could take on the command line. It added one named “quarterly” which directed the script to write in SoundExchange 2013 format. If the quarterly argument on not specified on the command line, then the output was raw icecast format. The change between version 1.0 and 1.1 of the script led to the defaults reverting, and this led to confusion between ibiblio staff and clients. In one support request ticket, a client asked that their report format be “changed back.” It was not clear to ibiblio staff at the time that the formatting had ever changed. It was determined at a later date that the format default was switched in the 1.1 script.

5.2 New System

Specifications for the new reporting system calls for a web-based application. As the ibiblio staff has previous knowledge with the PHP programming language, it was selected as the language for the new system. In addition to PHP, previous knowledge of the MySQL database management system (DBMS) led to it being selected as the database for the new system. The first step in designing the new system was to review what the clients were looking for.
One complaint we heard from clients regarding other systems was the lack of a single sign-on system. Clients wanted to be able to use one username and password to access all ibiblio services. Unfortunately, this type of system is not something that can be implemented quickly. We settled on using Google for authentication. It allows clients to use their existing Google account to access the new system.

We also decided that Python was still the best language for parsing the log files. PHP and Python can easily share data stored in the MySQL database. Fortunately, PHP also has a built-in function called “exec” that allows it to call an external program. This function will allow our new system to interface with the 1.1 script from the old system. Version 1.1 was ultimately selected since it covers the needs of both use cases discussed in section 4.1.

Finally, models of the new system were created before writing any code. Below is the database diagram:

![Database Model](image-url)

*Figure 3: Database Model*
A few user interface models were also created. The first (Figure 6) was for the report generating screen. A requirement of the system was to keep the workflow similar to the previous system. The initial model simply asks for the same information that was submitted in a help form under the previous system. Another screen (Figure 7) was called the previous reports screen. Based on feedback from clients, reports will be emailed once complete. The previous reports screen allows clients to download a file that was previously generated and emailed.

**Generate Report**

- **Station:** Enter Text
- **Format:** Enter Text
- **Start Date:** Enter Text
- **End Date:** Enter Text

![Generate Report UI Model](image)

*Figure 6: Generate Report UI Model*

**Reports:**

- Report run 2016/01/15
- Report run 2016/01/15
- Report run 2016/01/15

![Previous Reports](image)

*Figure 7: Previous Reports*
6. **Implementation**

As part of a security effort to keep public systems isolated at ibiblio, a new Virtual Machine (VM) was created to host the new system. CentOS 7, along with Apache 2.4 and PHP 5.6 was installed on the new VM. The VM also has its own public external Internet Protocol (IP) address, which allows clients to connect to it from over the Internet. A class for puppet was also created to build the configuration files for Apache. Puppet allows VMs to be destroyed and rebuilt with minimal user input.

After the VM was built, our first challenge was to build the user interface from our existing models. This was completed using HyperText Markup Language (HTML) and Cascading Style Sheets (CSS). After some initial testing, we also decided to employ Asynchronous JavaScript and XML (AJAX) in our new system. AJAX allows a webpage to stay loaded in the browser when requesting updated information. Once the new information is retrieved, only sections of the page is changed. This gives the user a better overall experience when interacting with the webpage.

Before we could start client testing with real data, we needed to implement our authentication layer from Google. This was accomplished by using the Google provided Identity Toolkit. This toolkit comes with everything you need to get your website authenticating using Google accounts. Although it allows multiple security providers, including Yahoo and Facebook, the decision was made to only offer Google accounts in the initial implementation.
Early internal testing showed a few minor issues that needed to be resolved by altering the original 1.1 version of the Python script migrated from the old system. For example, files uploaded to SoundExchange needed to be compressed by default. This meant adding the Gzip library to the script. An email feature was also created in PHP. This feature allows the generated log file to be emailed to the client once it is complete. This option was added so clients didn’t have to wait while a file was being parsed. The parsing process can take an extended amount of time, especially for larger stations.

The final user interface is shown in Figure 8. The user interface for the new system has all the features from our model, except the progress bar. It was determined that a progress bar was not feasible, due to usability issues. We didn’t want users to feel like they had to wait for a report to complete. The user interface alerts the client after a report has started. It explains how they will receive an email once the report is ready, allowing them to move on to other tasks in the meantime.

![Figure 8: Final User Interface](image)
7. Testing and Results

Following the RAD methodology, we kept in contact with users during the system implementation. This allowed us to get some valuable feedback. First, two of our three clients had confusion with the log reporting options. From the drop down menu, we offered the options “Raw” and “Triton.” Triton was a legacy name from the old system Python script. It was not clear which option was needed for SoundExchange. In response to this, we were able to quickly alter the site to show SoundExchange in the drop down menu of the user interface.

We also received feedback on the file compression used in the report generation. One client showed how the compressed file contained an absolute path with the original server folder structure. In order to access the uncompressed file, the client had to traverse a list of subfolders. We were able to quickly modify the code so the file was located in the root of the compressed file.

In addition to soliciting client feedback, we also looked at logs generated by Apache. These logs show us when a client logs in, if they run any reports, and if they download reports. We had one instance where Apache showed us an error that was not reported by the user. The problem, a permissions issue, was corrected and the logs were reviewed later to confirm the issue had been resolved.
During the testing phase of this project, the decision was made to build end-user documentation for the new system. An online documentation site was built for the new system using MkDocs, which is a website generator for product documentation. Although the three clients who were testing the new system had no issues learning it, we felt that documentation might be helpful to others, including new stations.

Finally, we received some positive feedback about our Google account integration used for authenticating. Clients appreciated having the ability to sign in with their existing Google account. Only one client didn’t have a Google account before trying the new system. However, it only took about 5 minutes to have them create one. Having the Google account integration has also decreased the setup time ibiblio staff needs to add a new user to the system.

Since the rollout of the new system, ibiblio has not received any support ticket requests regarding problems with report generation. In addition, no requests have come in to manually generate reports. This means ibiblio staff has not had run the Python script by hand to generate a report since the new system was put in place. Finally, based on timestamps in the log files, 79 reports have been generated by clients on the new system in 2016.

Overall, user feedback regarding the new system has been positive. One client said the new system was “[m]ore than greatly appreciated.” A fourth client, who was added to the final testing of the system, simply said it worked as expected. The lack of complaints appears to be due to the system being ready-at-hand. Users are unlikely to complain unless the system is failing in some noticeable way.
8. Limitations

Client testing showed a few minor limitations of the new system that can be addressed in the near future. For example, some users may prefer to disable the option to send the report as an email. We are also investigating the possibility of submitting the reports directly to NPR/DS. This would remove the step of having the client download the file and then upload it.

Another limitation is system administration. A separate webpage was built to allow ibiblio staff to add users and stations. The site operates correctly, but it does not have all the features needed to be considered complete. For example, user permissions cannot be altered without manually calling a Structured Query Language (SQL) command in MySQL. Radio stations also cannot be modified from the administration site.

This new system also fails to resolve a third use case that came up during testing. Some stations would like to have reports generated on a schedule. Currently, ibiblio has scripts that perform this task. They are called each night from cron, which is a Unix daemon that executes tasks according to a schedule. Although this process works, clients would like the ability to change their schedule through the new system webpage.
9. Future Direction

In addition to resolving the limitations listed in the previous section, the new system will also benefit from having providers other than Google added for login authentication. This would give clients the freedom to use whatever account type they prefer when logging on. Since this system uses the Identity Toolkit, adding other providers should be trivial to implement.

The hardware infrastructure example in this paper lacks any high availability (HA). HA allows systems to operate in the event of partial hardware failure. For example, if a server goes offline, the system would be able to operate on remaining online servers. On the example in Figure 1, a failure of the Icecast server would take all streams offline. An updated system is pictured in Appendix C.

As seen in Appendix C, implementing HA calls for two servers to act as routers, and at least two Icecast servers (“Linux Virtual Server,” n.d). Adding the hardware required for an HA system is quite simple. The difficulty is the software. A few software packages are available to users wanting HA in their infrastructure. Two of the most well-known packages are HAProxy and Linux Virtual Server (LVS).

However, for Icecast, HAProxy will not work. Reporting to SoundExchange requires the listener’s IP address from the Icecast logs. When Icecast is sitting behind HAProxy, it will always log the IP address of the HAProxy host. Although HAProxy offers a header called “x-forwarded-for” to pass the IP address of the client, Icecast does not support this header.
LVS has been successfully used to add HA to an Icecast cluster (Graakjær & Johnsen, 2008). LVS operates by having two systems acting as routers. One is labeled primary, while the other is labeled secondary. If the primary host goes offline, the secondary becomes primary. Removing the router as a single point of failure is what adds HA to the connection (Kopper, 2005) between multiple Wide Area Network (WAN) connections and the Icecast hosts.

Although LVS has been used to manage a cluster of Icecast hosts, there has not been a documented case of LVS being used to manage an Icecast cluster on an IP version 6 (IPv6) network. IPv6 was designed to be used without Network Address Translation (NAT). However, this design overlooks HA in IPv6. NAT is what allows LVS to handle HA on IPv4 networks.

A temporary solution to this problem on other services, such as FTP, has been to implement a round robin configuration for the host in Domain Name Service (DNS). However, this solution is not ideal for Icecast, since a server going offline will disrupt all streams that were connected to that host. In a round robin DNS configuration, the disconnected hosts won’t automatically reconnect.

As the availability of new IPv4 addresses has been exhausted, it will be critical to add IPv6 support in the near future. Ideally, all HA services, including Icecast, would be available on both IPv4 and IPv6 networks. Further work will need to be done in order to find a viable HA solution to provide IPv6 support in an Icecast cluster.

It is also recommended that ibiblio implement versioning control for all of its custom scripts and programs built in-house. Although larger ibiblio projects currently use “Git” for versioning control, small programs, such as the Python script migrated for the
new reporting system, do not. This, in addition to limited documentation, can lead to similar issues that came up when the Python script was updated from version 1.0 to 1.1. With a larger emphasis put on documentation, and added versioning control, future complications occurring with custom built software should be greatly limited.

Finally, following the SDLC (see Appendix D), the last step to complete is known as the system evolution. Although the new system is considered complete, it is not finished. Due to the changing nature of software, this system will need to be continually tested and updated. New requirements and computing technologies are likely to make the current version inoperable. This could include new SoundExchange formats or PHP upgrades.

No matter how the system evolves, one step of the process will never change: sitting down with the client. The client needs to be involved at the very beginning, starting with the planning and requirements phases of the SDLC. Not only will this give designers valuable input, it will also help bond the user to the new system. Project managers and system designers can have what they consider to be a perfect system. However, if the user refuses to adopt it, the system is a failure.

The live system for this project is located on the web at:  
https://radioreports.ibiblio.org

The documentation for the system is located on the web at:  
https://www.ibiblio.org/docs

The code for this project is hosted on the web at:  
https://github.com/dcowhig-ibiblio/radioreports
10. Appendix A

System Documentation
System Documentation, Continued

After you insert your Google account username, the system will forward you to Google to complete authentication.

Once you are authenticated, you will be redirected to the Radio Reporting Dashboard.
System Documentation, Continued

After the report has been generated, you will receive an automated message with a customized link to download.

If you do not get an email within 30 minutes, please let us know by submitting a support request. Be sure to include your username and reporting dates.

When ready to generate a report, the Biblio Radio Reporting System can be found at radiorpts.biblio.org.
System Documentation, Continued

**Log Formats**

ibibio currently offers reports in two formats: Raw and Triton. Use Raw to download the unaltered libcall log format. The Triton format (shown below) is required for SoundExchange reporting.

### Triton Format Example

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Date</th>
<th>Time</th>
<th>Stream</th>
<th>Duration</th>
<th>Status</th>
<th>Referrer</th>
</tr>
</thead>
</table>

### Compressed Files

Downloads from the ibibio radio reporting site are compressed by default. This saves bandwidth when transferring large reports. Our system compresses the files using gzip. The extension for gzip is “.gz.” If you do not have access to `gunzip` on Linux/OSX, other software packages available to extract files from a gzip are 7-Zip for Windows or The Unarchiver for OS X (Mac).

**Requesting Support**

If you have a question related to ibibio radio reporting, try consulting our [interactive Q&A site](https://example.com). Is the answer to your question not there? Log in and ask it! Someone in our community probably has the answer and will post it pronto! We also know that you’ve got a wealth of knowledge that could be useful to other ibibians. Please strengthen our support community by posting your own answers.

For account and/or password problems, please fill out the [help request form](https://example.com/issue). Any other technical question or problem report can be handled at the Q&A site.
## 11. Appendix B

### Copyright Duration of Musical Compositions

<table>
<thead>
<tr>
<th>Date of Work</th>
<th>Protected From</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Published before 1923</td>
<td>In Public Domain</td>
<td>None</td>
</tr>
<tr>
<td>Published from 1923 – 1963</td>
<td>When published with notice(^1)</td>
<td>28 years + could be renewed for 47 years, now extended by 20 years for a total renewal of 67 years. If not so renewed, now in public domain.</td>
</tr>
<tr>
<td>Published from 1964 – 1977</td>
<td>When published with notice</td>
<td>28 years for first term; now automatic extension of 67 years for second term</td>
</tr>
<tr>
<td>Created before 1-1-78 but not published</td>
<td>1-1-78, the effective date of the 1976 Act which eliminated common law copyright</td>
<td>Life + 70 years or 12-31-2002, whichever is greater</td>
</tr>
<tr>
<td>Created before 1-1-78; published between 1-1-78 and 12-31-2002</td>
<td>1-1-78, the effective date of the 1976 Act which preempts state common law copyright</td>
<td>Life + 70 years or 12-31-2047 whichever is greater</td>
</tr>
<tr>
<td>Created 1-1-78 or after</td>
<td>When work is fixed in tangible medium of expression</td>
<td>Life + 70 years(^2) (or if work of corporate authorship, the shorter of 95 years from publication, or 120 years from creation)(^1)</td>
</tr>
</tbody>
</table>

*Table 3: Copyright Duration (from Rose, 2002)*
12. Appendix C

System with High Availability

[Diagram of HA Webcasting Infrastructure]
13. Appendix D

Software Development Life Cycle (SDLC)
Bibliography


FTC. (2012). Enforcement Advisory No. 2012-02 FCC ENFORCEMENT ADVISORY CELL JAMMERS, GPS JAMMERS, and OTHER JAMMING DEVICES CONSUMER ALERT: Using or Importing Jammers is Illegal


