UWTV – Working With Online Media

By Andy Rose

Table of Contents

Working Atmosphere

The internship position that I worked for the summer took place at UWTV on the University of Washington campus. UWTV runs the television station for the school, but it is also the IT department. They merged some time back in order to have an increased budget. My first day on the job was spent being introduced to people and learning about the systems that they use. The first couple of hours were spent in a meeting, where I got to meet my boss, Laurens Banker, and my supervisor, Paul Marcontell. Laurens is the UWTV/IT Department Director and Paul is the IT coordinator. I also got to meet Jeff, who is the UI/UX specialist, though through the term of the internship we communication with him was minimal.

After the meeting, it was time to get situated into the workplace. I was given my own cubical to work in as well as a computer to conduct my work. After all of the hardware was set up, I received my own login credentials through the University. I also got a user account with the platform, which will be talked about later. The remainder of my first day was spent installing the necessary software and familiarizing myself with the systems and the different abilities it possessed.

System Information

During the initial meeting, I was given a brief overview of the systems in place at UWTV as well as the procedures in which activities are carried out. The main process that concerned the work that I would be doing involved streaming video and audio through their website. The system for online media streaming at UWTV begins with the "Platform", which they renamed to "**MediaAMP**". The MediaAMP is an online interface that organizes and ports the various media objects. It is what communicates with the servers in order to provide playback of media in its provided embedded media player. Over the course of this description, the different functions of the MediaAMP will be discussed. Below are the steps to creating media that can be streamed online. See also Appendix A for a systematic layout of the system.

1) **Planning** – The first step is to plan the production of the media. Though the IT department does not do the actual production, they still need to create a "**media object**" in the MediaAMP. This consists of declaring the name for the object and adding other attributes, such as a description and tags.
2) **Editing** – Though this is the next step in the procedure, the IT department is not responsible for the editing of any media.
3) **Encoding** – After the editing is finalized, it needs to be converted or "**encoded**" into the following file types. The IT department does not do the majority of the encoding, as there are programs set up to do this, but it is important because the MediaAMP needs to be able to recognize the encoding of each file in order to stream the media. Once the files are encoded, they are moved into the "**isilon**" storage server.
    a. Mezzanine (.avi) - This is the highest quality and can later be used to make other file types in good quality.
    b. Streaming High Quality (.mpeg) – This is used for streaming video to users that have a decent bandwidth.
    c. Streaming Low Quality (.mpeg) – This is also for streaming over the website, but for users with a slower connection.
    d. Podcast Video (.mp4) – Low resolution quality that is compatible with mobile devices.
    e. Podcast Audio (.mp3) – Audio playback compatible with mobile devices
4) **Backup** – All finalized media is backed up onto tapes that are stored in large rooms for later use. Tapes are used instead of hard drives because they are more reliable after sitting for long periods of time.
5) **MRSS** – At this point, it is time to link the stored media to the MediaAMP. To do this an **MRSS** (Media RSS) script must be created that includes the location and specific information of each file that needs to be routed to the MediaAMP. Once the script is completed, the file containing the script is moved to the "**Watch Folder**" which is checked by the MediaAMP periodically.
6) **Ingest** – Once the MediaAMP recognizes the MRSS file, it "ingests" the media, meaning that it takes the information in the script and uses it to add the new media items into the media object created earlier, as well as link the media items directly to their files residing in the Isilon holding

server. This also adds all of the important information attributes to each media file within the media object.

7) **Thumbnails** – This step can be performed simultaneously with the media, but needs to be distinguished as it is a different process. There are two sizes for the thumbnails, which are displayed on the website to the user. The large thumbnail is for the embedded media player, before the user pushes play. The smaller thumbnail is used for links on other pages that lead to the page containing the media. As with the media items, an MRSS script is generated for the thumbnails and ingested by the MediaAMP. The key difference is that MediaAMP physically stores the thumbnail files. This is so that communication to the server isn't bogged down by users browsing through pages containing a large number of links to videos.

MRSS Generator Project

The task presented to me during my first day of work was automating the step for generating the MRSS script. As discussed earlier, MRSS needs to be generated for all media items and thumbnail items. Being intensely understaffed, this time consuming process looms quite the burden for those in the IT department. Having a system to automate the process with a few clicks would greatly reduce the workload of employees as well as increase the speed of the whole process. The project was not given to me as a whole at first. In fact, I thought it would be a matter of hours before I finished the task. With this mindset, I rushed directly into the project with no planning.

From this situation, I learned that it is important to ask as many questions as possible, even when a job seems simple because it will save you a lot of work in the end. If I had known the project would be so extensive, I would have written out a plan in order to increase my efficiency in the creation of the program. Instead, I was asked to add one feature after another and had to refractor three times to get the program to run smoothly.

Thumbnail Script

The first task that I was asked to complete was building a program that would generate an MRSS file for ingesting thumbnails into the MediaAMP. I decided to create the program in C# because it is the language I am most familiar with and it is extremely fast to make a GUI. Since my experience with RSS feeds was fairly small, I started by looking over previous MRSS scripts for this function in order to understand how the script was written. Once I got a good grasp of this, I started building the program. The program was a simple form that allowed the user to insert the "**Media ID**" of media object that exists on the MediaAmp and select the two thumbnail files using a file browser. From here the user could build the script, view it, and send it to the watch folder.

There were several complications that I encountered with this portion of the project. The first situation I examined was that there was no way to retrieve the information on an object directly from the MediaAMP suite. The MediaAMP runs on javascript in the browser and therefore contains no object information for its contents in HTML. This means that the user will have to copy and paste the Media ID into the program from the suite. There was no workaround for this that existed in the scope of the project. This problem leads into the second issue that I ran into, which was more of an inconvenience issue for the user. When working with the MediaAMP on a browser, it is frustrating having to switch between programs. Since the program takes so little room. My solution was to set the program to stay on top of all other windows.

I had a working version of the program completed in a matter of two days. I do not think this was expected because my supervisor, Paul, was shocked I had completed it so fast. This may have contributed to the massive increase in the desired functionality of the program, stemming from the excitement of how fast it was to make it and how much time it would save. Though many new features were requested at this time, I began implementing the function to generate script for media items into the program.

Media Script

I initially created the media portion of the program the same way I created the thumbnail portion by allowing the user to set the Media ID and select the files. Concerned about making the form too bulky, I added tabs so the user could switch between media and thumbnails. I hardcoded all of the script from the examples provided to me and used the file information to complete the script. Again, the process worked and only took a day to create. See Appendix B for the final view of the main form.

Up until this point, the program had been getting local files that were stored on the machine and ingesting into a MediaAMP that was created for testing. This was due to an issue at the beginning in giving my account permissions to access files on the protected server. Now that I had a working version of the program that could build proper script, it was time to start testing using real data. This is where problems began to arise. The first encounter I faced was that the C# file browser couldn't search through network locations. This was solved by mapping to the network drive outside of the program. Other solutions were thoroughly analyzed, but mapping was the only one within the scope of the project. Once the user had access to the network server, another problem arose. The file browser does not get the full path name of files on a mapped drive. After hours of online research, I found a wild back-door solution that worked. At this point, the user could successfully generate script from the network folders.

Added Features

With the program working, it was time to start fulfilling the requests to add more features. The first addition was the most crucial, which was maintaining the "**Server ID**" and the "**Transform ID**" for each file type. Though these were hardcoded before, it is something that the user would need to be able to change. This goes along with another requested feature that came later to be able to add, remove and alter the media "**Assets**", which are the different media types. Each asset has a unique Server ID, used to call to the location of the destination of the file, and Transform ID, used to decipher the encoding of the media. After several iterations I ended up creating a database that held all of the information on all of the assets, carefully selecting my attributes to include those needed for MRSS development, as well as those that might be used eventually to communicate directly with the MediaAMP. The settings for the IDs would most likely need to be changed more often than the assets, so I decided to include them in the main form under the file menu tool strip. I inserted text boxes directly into the menu for the user's convenience, and to minimize the number of forms that the program could shell out (Appendix B). I also created a secondary form that the user could access to make changes to the assets (Appendix C). These features are important because it is hard to tell what the future for UWTV will hold. From this, I learned the importance of programming for possible situations that might arise later instead of just the here and now.

It was while implementing the asset features that, for the first time since I started using it, I found functions that I really disliked in Visual Studio 2010. The first of these was using their built-in database development tools. In the past I have always used Access, but in the spirit of learning new things in my internship and maintaining the program as one solid unit, I decided make a built-in database. Building the database was not as much difficult as it was time consuming from having to build

it using SQL queries. However, it was the implementation that I found especially frustrating. I attempted to build the table for altering the assets with a DataGridView object in coherence with the database. The idea seemed simple enough, but changing the settings on the object to work with my data proved to be an extreme setback. After two days of battling with the object, I finally got it to act the way that I wanted it to. However, what really put the nail in the coffin for me was that, after all of the work I did to get it to work, I could not get the changes to save in the database and stay after the program restarts. I researched solutions and tried a dozen different things, but something to do with the permissions in the solution folder and the communication between the object and the database connection would not allow me to continue. Frustrated for wasting so much time, it tore it all apart to build and implement an access database, which was working in a matter of hours.

The rest of the features concerned the programs user-convenience and aesthetic appeal. Such features included settings for keeping the program on top of other windows, which tab should be selected on startup, where the default locations of the different files would be, adding one or many files at a time, etc. While researching ways to save user settings with their login account, I found that Visual Studio has a great feature to build settings from within the IDE that also works with the windows account. Using this was extremely easy and efficient to use. Other important extra features include copying a backup of each generated file to a backup location and a fully descriptive, custom, in-depth help page (Appendix D)

Undeveloped Features

As mentioned several times throughout this section, there are many features that were out of the scope of the project. This was mainly due to the fact that the internship was only 160 hours. In order to create an effective program in its entirety, this is not a lot of time. The most convenient feature that could have taken the program to the next step is incorporating the program directly into the web-based Platform (MediaAMP). Developing a web app that wouldn't need to be a separate program would have substantially easier for the user to work with. This was out of the scope for two reasons, the first being that I personally have not worked with developing web-based applications, so learning to do this and making an effective program would have taken too much time. Secondly, the MediaAMP suite is programmed in Java and used in the browser through an API. The system has options to add extra features and systems using web-apps, but you have to generate them using their system. The other barricade with this was that all of the information for the objects is not stored in HTML, but instead the Java program running through the API and plug-ins, making any data within the program inaccessible to any outside web-app. Another potential feature includes giving the program access rights and privileges to the server locations. This would make the program accessible to those outside the IT department, allowing those working in IT to micro-manage the workload to a wider range of people.

Learning Experience

The main purpose of being involved in an internship is to get real-world workplace experience. It is good practice toward working with others in a working environment outside college. I feel that participating in this internship has significantly helped to prepare me for life after college. The experiences that I took from this internship cannot be taught in the classroom, which is why I feel it is important to absorb as much from it as possible.

Key Concepts Learned

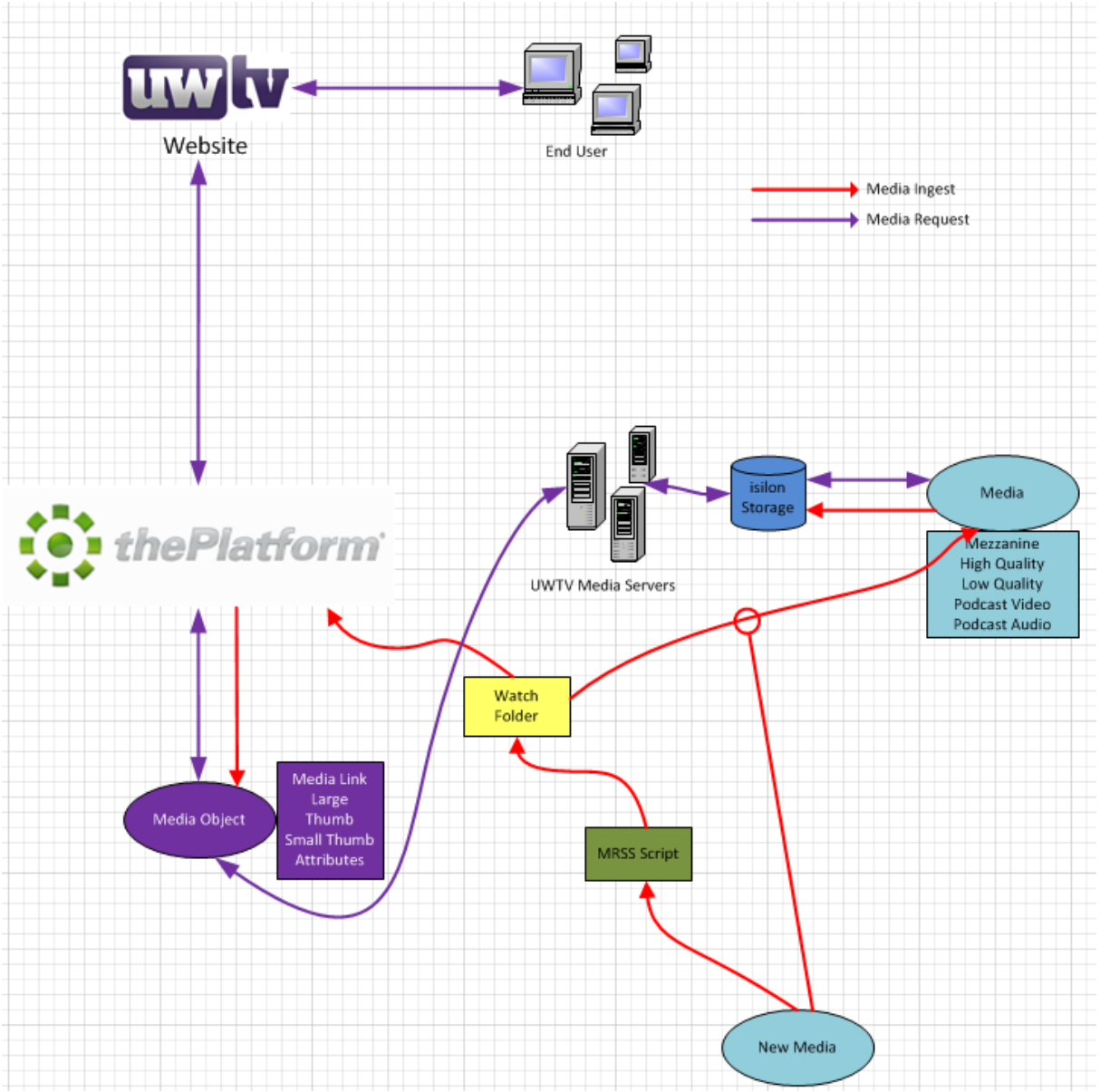
Of the main values that I will take away from this experience, I believe that communication is the most important. During the course of my internship, I worked mainly under two people. However, throughout almost the entire time, one of them was out of the office either on vacation or working remotely. When others are working outside of the office, it is extremely important to keep them up-to-date with any project that they are involved in with you. Even if they are on vacation and don't reply to an email, most supervisors still appreciate knowing what the status of the project is. Not only does it allow them to address any urgent concerns, but it also helps to build confidence and trust in whoever is working on the project.

The second most important key to success I learned at this internship is to look to the future. Users are much more intuitive than they used to be, so it is important to allow them as much functionality to change and set the program to work how they want it to. This was especially important for this particular program because the user-base consists of all computer-savvy IT workers. During my education in college, I never learned how to generate user settings and provide functionality to change the actual program from the user level. I learned that when creating a program, it is important to expect changes in any attributes, variables or objects. Users expect to have settings. I discovered that a program needs to be able to do react to a wide range of users having different computing styles. An example of this would be copying and pasting the ID into the text box. Some users may use the context menu, others may use keyboard macros, and others may even try to click and drag the text into the box. All of these need to be properly planned for. The best way of doing this is to either have the program tested by multiple users, or simply imagine as many variations as possible of every function.

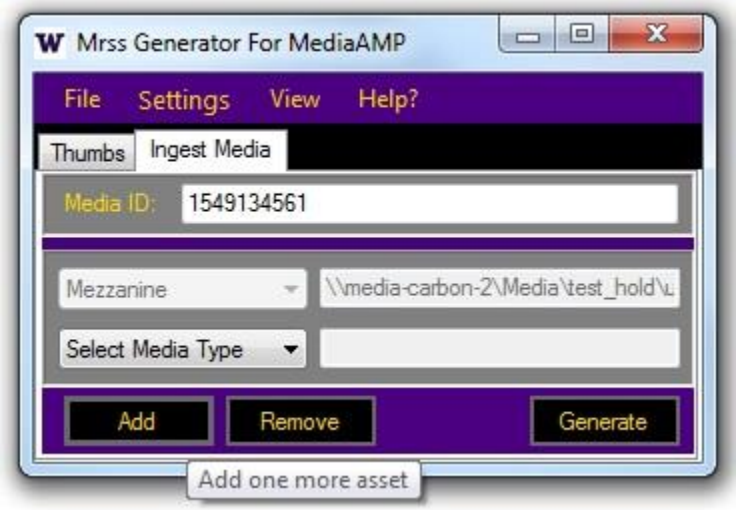
Another major aspect of working in the real world is how extremely important it is to ask the right questions. This is something that I personally struggle with. I am the kind of person that starts thinking of ideas immediately after being told about the project. At this point I normally write all of my ideas down in order to analyze and organize my thoughts, which is positive. However, my downfall is that during the brainstorming process I tend to get overly excited to start the program, leading to me jumping right into the programming part. It is here that I need to look over what I have come up with and compare it to the requirements to find any holes of missing information or requirements that the supervisor may have missed. Not doing this before creating the program led to me having to refractor several times and reformulate much of the code which I had already written. In the future, I plan to come up with a set of at least ten intuitive, informative, and thought-provoking questions to ask my supervisor.

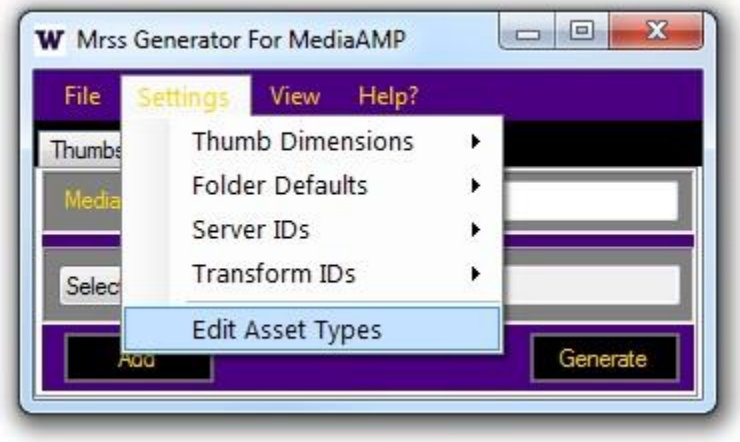
The last bit of knowledge that I gained was the importance of properly commenting code. This is actually something that I have always been very good at. I am happy to be in one of the only professions where chronically explaining everything is looked at as a positive attribute. The reason that commenting is so important is so that others can understand how the code works. It is naïve to assume that your career in one location will outlive your code, just as it is immature to think that others who can't read your code have no business looking at it. As stated above, it is important to consider the future. Commenting the code allows others to understand how the program works if they ever want to add features or change the program. It also allows for easier navigation for the programmer while creating a program. Lastly, when working with a group on a project, it is important that everyone understands exactly how the code is working in order to improve it or build added features using that section.

Appendix A – System Diagram

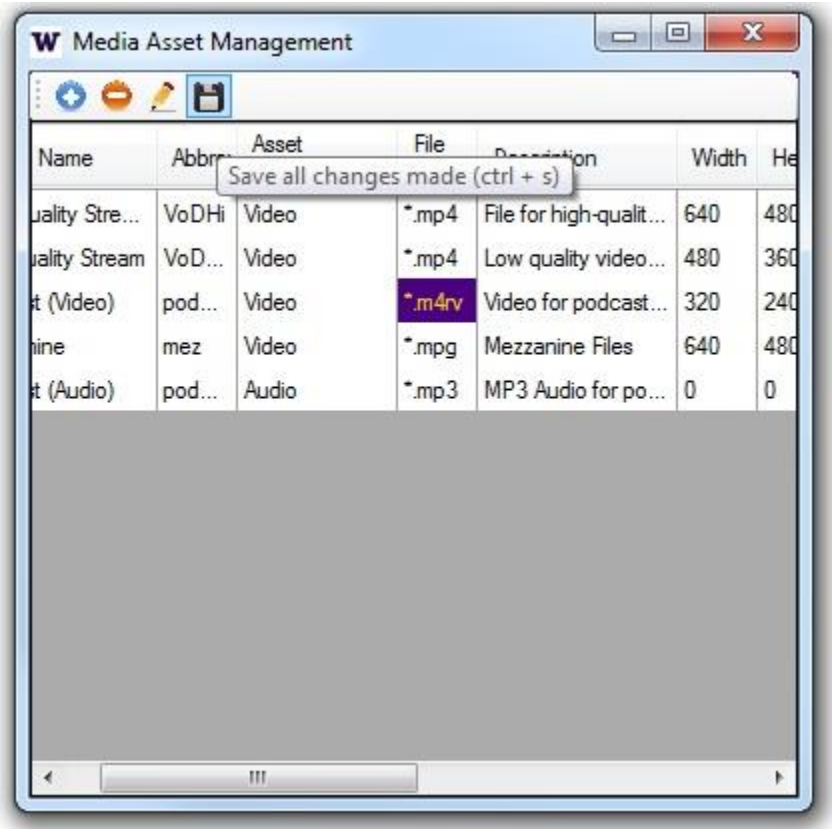


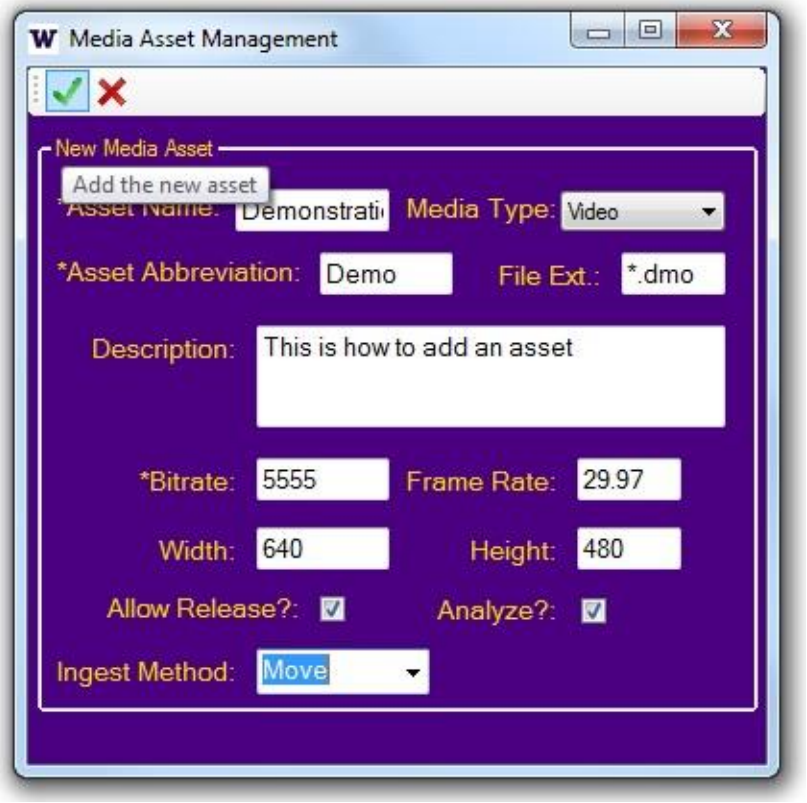
Appendix B – Main Form





Appendix C – Asset Maintenance Form





Appendix D – Help Form