

A Video Manager

Nicholas Omann

4/19/2009

Abstract

The goal is to create an open source project known as A Video Manager (AVM), which is a fully functional video management system that allows a user to easily navigate their video collection when it is stored on their computer, on YouTube, on Hulu, or on DVD. In addition to the AVM application, there is also a project site complete with bug tracking, beta releases and version control. Documentation is available for both users and developers interested in joining the AVM Project in future releases or those merely interested in expanding AVM for their own use. The source code is readily available for any developer. It is licensed under the lenient MIT/X11 License and hosted at <http://www.codeplex.com/avm>.

Reason for the Software

Similar Software

There are many existing applications that allow users to view video files. In fact, there is a video player shipped with every operating system. Windows provides Windows Media Player (Microsoft, 2009), Macintosh OSX has QuickTime (Apple, 2009) and a Linux based operating system generally offers MPlayer. While these programs are very good at playing videos, they all lack a key feature. None of the programs do a very effective job of keeping track of video files and making them easy to browse. They either expect users to organize their video files via the file system (MPlayer and Quicktime) or provide file organization for a specific file format (Windows Media Player). Thus, these media players are lacking the major feature of AVM.

There is software designed to browse a video collection, but many of these programs do not have a simple user interface and they generally are not designed to be used on a computer, but rather, they are aimed at the media center market. The most widely available product is Windows Media Center, since it comes on almost every computer running Windows Vista (Microsoft, 2009). A user is allowed to keep track of video files, but when looking through the collection in Windows Media Center the user can view them only as a screenshot. This is a problem because of the way Windows Media Center takes thumbnails. When Windows Media Center takes a thumbnail it grabs a frame that is a specified number of seconds into the video. When looking at a folder containing 12 episodes from a series, the same thumbnail tends to appear for every video. This makes it harder to recognize an episode and thus makes the organization confusing. Windows Media Center file organization is also completely dependent on file hierarchy, which can be a problem for people who do not keep track of how they store videos on their computer. Another missing feature from Windows Media Center is a way to look up videos, because it is designed to be used with a remote. Most media center programs have these same problems, and though some are able to keep track of DVDs, most do not have a simple search function. All require visually recognizing a video, which works well on a TV but isn't very helpful when watching videos on a computer with a keyboard.

Influences from other Software

AVM has been influenced in many ways by other applications. Unix programs were designed to do one thing really well. AVM is also designed to do one thing very well as AVM just manages videos, it does not play them. As in Unix where output from one application can be piped to another application, AVM offloads video playback to either a web browser or to the user's default video player. These philosophies from Unix have had a large influence on the architecture of AVM.

The user interface layout for AVM is similar to those presented by the media players Amarok and Exaile (Amarok, 2009; Olsen, 2009). Amarok is one of the most widely used media players for GNU/Linux desktops. Both Amarok and Exaile have a tree on the left side to setup a playlist and the playlist is shown in the middle to right side of the application window. A similar layout is used in AVM. The playlist area in AVM is actually just the list of the videos in the "Group" that is selected on the left side because in AVM there is no need for a playlist (most people watch just one video at a time). The left hand side is not a tree view; it is a design that facilitates working with video more effectively. The user can go back in the hierarchy or select a folder and go forward into it. The divergence from the way that Amarok and Exaile handle their left side list is due to the underlying difference in the way that the list is formed. In Amarok and Exaile this list is formed using metadata from the music that, for the most part, stays consistent. In AVM the group list is user generated and entries are more often created and removed from this list, so it needs to be more dynamic. Also, the original attempts to use a tree resulted in confusing layouts.

The Need for A new Video Manager

There are a variety of reasons why there is a need for a video manager built in the style of AVM. AVM offers useful options that are missing from many current video players or managers.

AVM adds better search functionality. This is a major feature that does not exist in most current video players and media centers. Many video players have ways to look through a library, but few have search functionality that works well when searching for videos. Though most applications in this area may have some way to search, few of them focus on searching as a fundamental tool. For example, Windows Media Center has a search feature, but it is hidden and does unexpected things, like finding shows on TV but not videos on a computer. This is not helpful since the TV show must first be recorded in order to take advantage of the search. iTunes has problems searching for videos as well (Apple, 2009). The iTunes problem, though, is caused by the limited scope of files that iTunes is able to play. Since most standard video files cannot be played on iTunes, most video libraries under iTunes are not searchable.

AVM adds metadata for videos. Unlike normal video players that handle just the video files, AVM provides a way to keep information about the videos. The only common video container that currently allows metadata is Microsoft's Windows Media Video (wmv) format (Microsoft, 2008). There are many other formats but few of them come from sources that provide adequate amounts of metadata. This can cause problems if a user wants to read the episode name when looking for a specific episode in a series. If there is no device for metadata in the file, additional information about the file cannot be stored. AVM provides a way to add metadata to video files to make browsing through videos easier. The data though is not stored in the file like it is for Windows Media Video files.

AVM allows Digital Rights Management (DRM) to be less of a hassle. For most people DRM just gets in the way. To view a video with DRM it is often necessary to use a specific application to play a specific video file. This requires more work because a user cannot view all files in the same software, and the software needed to play DRM files often lacks features like video organization. AVM will keep track of video files and, when a file is played, the proper application will be started. The user is not limited to just the application that plays a file in order to keep track of it.

Lastly, AVM allows the user to track the episode that was last viewed. This is very important when watching a complete series of TV shows. After watching a few episodes of one show, a person could watch another series and then return to the first series at the point of departure. With AVM there is no problem returning to a previously watched series, since the last episode watched stays highlighted.

AVM clearly offers useful features that are different from similar applications. Like other video related applications, AVM is also specialized. AVM's specialty is keeping track of a large library of video files, YouTube/Hulu links, and DVDs.

The Whys

Why use C#/.NET

The choice of a programming platform is one of the most important decisions for a project. There are several good reasons why C#/.NET, a major software platform, is well-suited for developing AVM.

The first reason for choosing C#/.NET was the availability of powerful development tools. Using Visual Studio 2008 with C#/.NET enables easier and faster development of AVM (Microsoft, 2009). The graphical user interface (GUI) creation is easy in Visual Studio thanks to WinForms, a .NET programming feature that allows the programmer to use a simple drag and drop interface to layout the way that an application will look to a user. This is much better than the older way of testing user interfaces, which was mostly a long and complex process of trial and error. Also, debugging is considerably easier in Visual Studio than in other integrated development environments. The ease of debugging is an important factor because the many lists used by AVM store huge quantities of data and, without being able to easily traverse these lists during development, it is very difficult to make sure that data is being passed properly.

Another reason C#/.NET was chosen is that it is well-supported on multiple operating systems. AVM can be developed once in Windows using Visual Studio and then the source code can be copied and used again in Linux or Mac OSX using Mono 2.0, which supports WinForms (Novell). Interestingly, MonoDevelop (the integrated development environment (IDE) for .NET via Mono) actually provided warnings for potential problems that Visual Studio ignored and this helped with code cleanup later in the project (Novell, 2009).

C# was the language chosen because it has much in common syntactically with Java and C++ and seems to offer the best of both languages. .NET provides very useful libraries that are similar to Java

libraries. The combination of C# and .NET is a more powerful platform than the others that were considered. Pure Java does not have a well-refined IDE and it has slower runtime performance (Sun Microsystems, 1994-2009). C++ using QT for libraries and for the GUI was a possibility, but learning to use QT from scratch is time-consuming (Nokia, 2009). C# uses WinForms, which makes it much easier to build a good user interface. WinForms also looks good on all the platforms for which it is available. Java on many platforms does not look good and it also lacks some of the more advanced controls that WinForms provides.

Why use SQLite

For this project the use of SQL was almost a given. The key purpose of AVM is to make it easy to keep track of all the videos that a person might want to watch. One of the key features of AVM is the ability to search through all of the saved videos. SQL makes this search easy and enables storing data in a way that other programs can utilize if they want to tie into the same database. The most important reasons for choosing SQLite were its size and interoperability, the license it is under, and its ease of use (SQLite Home Page, 2009).

Size and interoperability were key factors in the choice of SQL databases. The two major options considered at the start of this project were MS SQL and SQLite. Microsoft's SQL Server product has a light version designed for being redistributed with applications (Microsoft, 2009). The only problems are that even the compact version of Microsoft SQL Server is still 1MB in size and it would require a separate installation from AVM. Ideally, the user should not need to install any additional software to support AVM. This is one reason that Microsoft SQL Server Compact Edition was deemed inadequate. Also, Microsoft makes Microsoft SQL Server Compact Edition only for Windows and so its use would require building two database backends to interoperate between different operating systems. SQLite, on the other hand, is a small integratable version of SQL (.8 MB). This is good because the installer can remain small, preferably less than 1 megabyte. Another major advantage is that SQLite is designed to work on any operating system, making it easier for AVM to be multiplatform. The SQLite wrapper that was chosen is System.Data.SQLite, a fully developed ADO.net provider (System.Data.SQLite). System.Data.SQLite works well in Linux and Windows.

The license of the SQL provider was also important in the choice of providers. Once again Microsoft SQL Server was not a good choice because it has a closed source license, which is contrary to the philosophy of creating an open source project. SQLite is public domain software. The C# SQLite wrapper is also in the public domain.

The last major reason for choosing SQLite was its simplicity. Unlike full-featured SQL databases, SQLite handles a much smaller variety of data types. Most of the data used by AVM are strings that are easily stored in SQLite TEXT variables and the size of the database variable properly expands and shrinks. Even though SQLite limits the number of data types available to the developer, the ADO.net provider that was used for AVM has an API that looks the same as a complete SQL database and can be programmed to act like a standard ADO.net connection. The ease of using ADO.net and SQLite facilitated the development of AVM.

Why be multiplatform

Writing a program for just Microsoft Windows ignores the 10% of users who use Apple's OSX or Linux (Net Applications, 2009). Software appeals to a larger audience when it is available to the Mac and Linux communities as well as Windows users. As pointed out earlier, this was one of the reasons for choosing .NET. There were three major options considered: Java, .NET, and QT. Java applications tend to look ugly on every system, though Java runs fairly well on all three major operating systems. .NET, on the other hand, uses native toolkits to create its user interface and it has the same libraries as Java. QT does not have a standard library as extensive as .NET and Java.

Why the MIT/X11 License

One of the main goals of this project is to make a video manager that is easy to support, modify, and extend. A key way to meet this goal is to have a very open license for the source code. Thus, the license choices were limited mostly to GPLv3 and BSD style licenses, which are the two most common open source licenses (Open Source Initiative, 2006). The biggest problem with the GPLv3 license is that it has language severely limiting the use of modified source code. Those who modify the source code of AVM should not be forced to license it under any specific license even if this means that they are not giving back to the project. BSD style licenses have few limitations as long as the copyright notice is retained. From the many versions of BSD style licenses, the MIT/X11 license was chosen for AVM. The actual BSD license has 3 versions (4, 3, and 2 clause versions) on the Internet. The difference between the BSD licenses is how they handle advertising. This can be confusing because developers do not know exactly what they can do with the code when it comes to advertising. The different versions can easily get confused as they are all commonly referred to as the BSD license. The MIT/X11 license does not address advertising because there is no inherent right to advertising that the BSD license was protecting against. At the same time the MIT/X11 license shares the same ideology as the BSD license but it is much clearer regarding what can be done with the source code. The full MIT/X11 License for AVM is in Appendix A.

Why this interface

As explained earlier, the basic layout of AVM was inspired by the Amarok and Exaile media players. The major goal in the overall interface layout was to create a very simple and easy to understand layout. This started by dividing up AVM into two distinct sections: the group section on the left and the video section on the right. There is a vertical divider between the two sections to allow for adjusting their sizes. The reason for the division is to keep all of the group controls separate from the video controls to avoid confusing the user. Each button (with the exception of the arrow buttons) is labeled to describe its purpose. The one button that may cause some confusion is the "Import" button, but it will not take long for a user to figure out its purpose. Another important thing to notice is that the search boxes are in opposite corners of the interface to minimize confusion about their purpose. The breadcrumbs that are shown above the video list are provided to help prevent a user from getting lost in the group hierarchy, which happened occasionally during testing without the breadcrumbs. If users do find some of the controls to be complex, there is a complete user manual that includes information about the function of every button.

The most complex control in AVM is the group explorer on the group side of the AVM interface. The group explorer control has two arrows that may be confusing. Clicking on a group enables the forward (“>”) arrow and allows the user to go forward into the selected group. If the user goes into the group then the user will notice they are now in an empty group and that only the back (“<”) button is now enabled. At the same time the user will see the breadcrumbs have changed. From this, most users should be able to figure out how to use the control without the aid of the manual.

The Process

Version Control and Bug Tracking

For the first part of the project, Microsoft’s Live Mesh product was used for version control (Microsoft, 2008). It allowed for the source code to be synchronized between the two computers that were most often used to develop AVM. Live Mesh synchronized the source tree between the computers. However, there was a slight side effect because synchronization shares folders and everything in the folders. Problems occurred if others wanted to use the mesh synchronized folders, since some files were accessible to only the user that owned them. This could be circumvented by copying the source code out of the synchronized folder and then deleting the locked files. This was more complex than needed and so it could not be easily used if more than one person wants to work on the project. Each person that wanted to view the source code either needed to be given a copy of the source code or be permitted access to the Mesh profile it was saved in. This method of synchronization was used only in the original development of AVM for personal backup and ease of access rather than as a tool to share the code with other users. Read-only access to the source code was permitted for those running the AVM application merely to look at the user interface.

Once the first beta version was ready, it was time to move to an open source project hosting site. CodePlex was chosen as the host for several reasons (Microsoft, 2009). First, CodePlex is the main site for open source .NET-based projects on the Internet. Second, it allows Subversion (SVN) and Microsoft’s Visual Studio Team System access to the source code that is stored for the project. Subversion is a well-known version control system and so it is a good choice for version control. Also, since it is supported on Windows, OSX and Linux, use of Subversion is consistent with the multiplatform philosophy. The third benefit to using Codeplex is that there are no ads. Sites like Sourceforge have many ads making it very annoying for users to cope with the advertising while trying to make bug reports and download the latest release (SourceForge, 2009). Overall, Codeplex fits the needs of AVM when it comes to project hosting and it provides a pleasant ad-free environment.

Since the move to Codeplex, all source control has been handled using TortoiseSVN (CollabNet, 2008). TortoiseSVN is a Windows application that allows for easier control of a SVN repository by integrating SVN into Windows Explorer. This enables using more than one computer for development activities and it also allows simple access to the source code. The “Source Code” section of the Codeplex site for AVM allows users to download any revision since AVM was put onto Codeplex so that interested parties can track recent changes. Codeplex also has a built-in bug tracking system and TortoiseSVN can

be configured to match up commits with issue numbers from the bug trackers, enabling those who are watching the development to see which bug fixes have been completed in a revision.

Beta Testing

Unfortunately, getting people interested in the project is not as easy as just putting the source code on the web. After the AVM Beta 1 release there were a few groups interested enough to download it. At this writing, both AVM Beta 1 and AVM Beta 2 had 36 downloads each. The AVM Beta 3 release at the time of this writing has 49 downloads. This is not bad considering that it is hard to get new projects noticed. Three factors seem to be encouraging downloads. First, the main page of Codeplex shows new releases. Second, there is word of mouth. Finally, blog postings were made after each release and can be found at Alpha's Place¹. However, few bug reports have been received. So far there has been only one beta tester. He created most of the 7 bug reports that have been filed on the Codeplex project site. All of the bugs reported so far have been fixed in the source tree. This does not mean that all fixes have been released. Currently one bug fix is waiting for the final 1.0 release of AVM. The small number of bug reports does not necessarily indicate a lack of interest. It could mean there are either no obvious bugs or none bothersome enough for the users to report it.

A Virtual Machine using VirtualBox was created to test AVM running within Linux (Sun Microsystems, 2009). OpenSUSE was the Linux distribution chosen because Mono is best supported under openSUSE (Novell, 2008). After downloading the managed code wrapper for the System.Data.SQLite library, the same ADO.net wrapper library used by AVM in Windows was downloaded and used to compile AVM without any codebase changes. So far, the Linux version of AVM seems to work, though not much testing of this version has been done yet. The focus has been on getting AVM working properly in Windows and, if Mono does the job it claims, AVM should run fine in Windows and on Mono. Patch information for the Linux version of AVM could be made available if needed.

¹ <http://alphacluster.wordpress.com/category/avm/>

Bibliography

Amarok. (2009). *Amarok*. Retrieved March 2, 2009, from KDE: <http://amarok.kde.org/>

Apple. (2009). *iTunes*. Retrieved March 2, 2009, from Apple: <http://www.apple.com/itunes/>

Apple. (2009). *QuickTime*. Retrieved March 2, 2009, from Apple: <http://www.apple.com/quicktime/>

CollabNet. (2008). *TortoiseSVN*. Retrieved March 2, 2009, from Tigris.org: <http://tortoisesvn.tigris.org/>

Microsoft. (2009). *Codeplex - Open Source Project Hosting*. Retrieved 2009 2, March, from Codeplex: <http://www.codeplex.com/>

Microsoft. (2008, May 12). *Information about the Multimedia file types that Windows Media Player supports*. Retrieved March 2, 2009, from Microsoft Help and Support: <http://support.microsoft.com/kb/316992>

Microsoft. (2008). *Live Mesh Beta*. Retrieved March 2, 2009, from Live Mesh: <https://www.mesh.com/Welcome/default.aspx>

Microsoft. (2009). *Microsoft SQL Server 2008: Compact*. Retrieved March 2, 2009, from Microsoft SQL Server 2008: <http://www.microsoft.com/sqlserver/2008/en/us/compact.aspx>

Microsoft. (2009). *Microsoft Visual Studio 2008*. Retrieved March 2, 2009, from Microsoft: <http://www.microsoft.com/visualstudio/en-us/default.mspx>

Microsoft. (2009). *Windows Media Center*. Retrieved March 2, 2009, from Windows: <http://www.microsoft.com/windows/windows-media-center/default.aspx>

Microsoft. (2009). *Windows Media Player*. Retrieved March 2, 2009, from Microsoft Media: <http://www.microsoft.com/windows/windowsmedia/player/default.aspx>

Net Applications. (2009, March 3). *Operating system market share*. Retrieved March 3, 2009, from Market Share by Net Applications: <http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8>

Nokia. (2009). *Qt - a cross-platform application and UI framework*. Retrieved March 2, 2009, from Qt Software: <http://www.qtsoftware.com/>

Novell. (n.d.). *Main Page - Mono*. Retrieved March 2, 2009, from Mono: http://www.mono-project.com/Main_Page

Novell. (2009, February 18). *Main Page - MonoDevelop*. Retrieved March 2009, 2009, from MonoDevelop: http://monodevelop.com/Main_Page

Novell. (2008). *openSUSE.org*. Retrieved March 23, 2009, from openSUSE.org: <http://www.opensuse.org/en/>

Olsen, A. (2009). *Exaile / Music Player for GTK+*. Retrieved March 2, 2009, from Exaile: <http://exaile.org/>

Open Source Initiative. (2006, September 19). *Open Source Licenses by Category*. Retrieved March 2, 2009, from Open Source Initiative: <http://www.opensource.org/licenses/category>

SourceForge. (2009). *SourceForge.net: Open Source Software*. Retrieved March 2, 2009, from SourceForge.net: <http://sourceforge.net/>

SQLite Home Page. (2009, February 4). Retrieved March 2, 2009, from SQLite: <http://www.hwaci.com/sw/sqlite/>

Sun Microsystems. (1994-2009). *Developer Resources for Java Technology*. Retrieved March 23, 2009, from Sun Developer Network: <http://java.sun.com/>

Sun Microsystems. (2009). *VirtualBox*. Retrieved March 23, 2009, from VirtualBox: <http://www.virtualbox.org/>

System.Data.SQLite. (n.d.). Retrieved March 2, 2009, from <http://sqlite.phxsoftware.com/>

Appendix A

The MIT License

Copyright (c) 2008-2009 Nicholas Omann

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

AVM User Manual



Nicholas Omann

v1.0

At this point AVM should look like Figure 2.

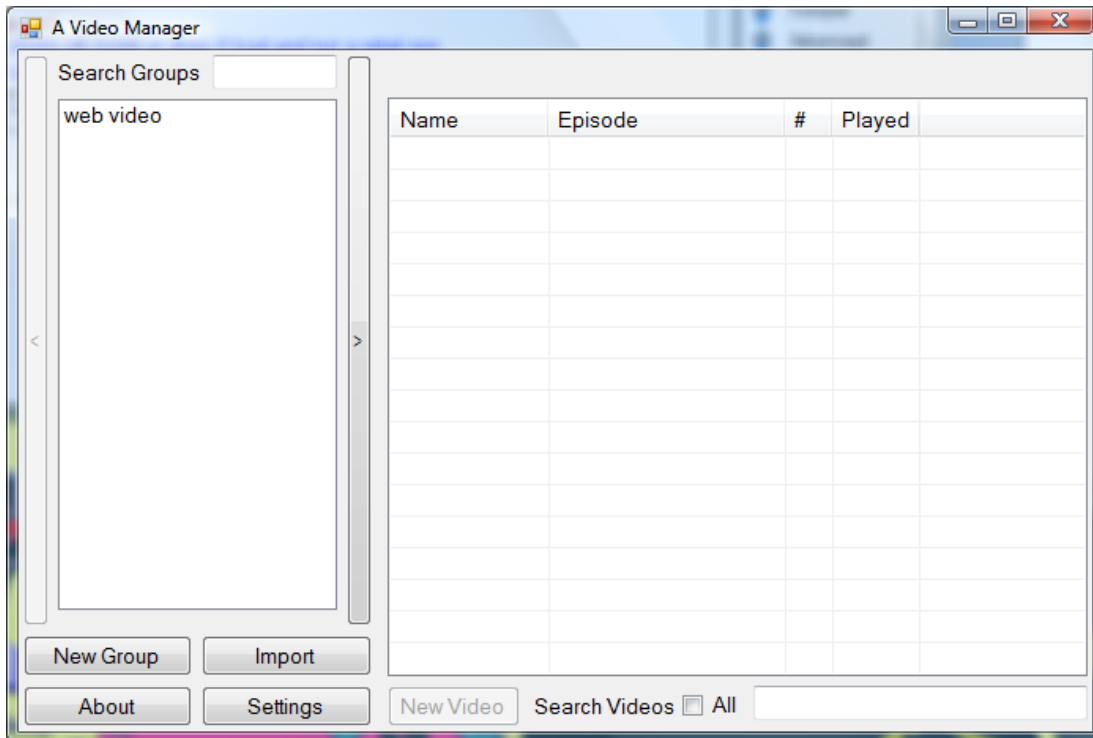


Figure 2

In order to add a video to the “web videos” group you must select the group and then press the **New Video** button as shown in Figure 3.

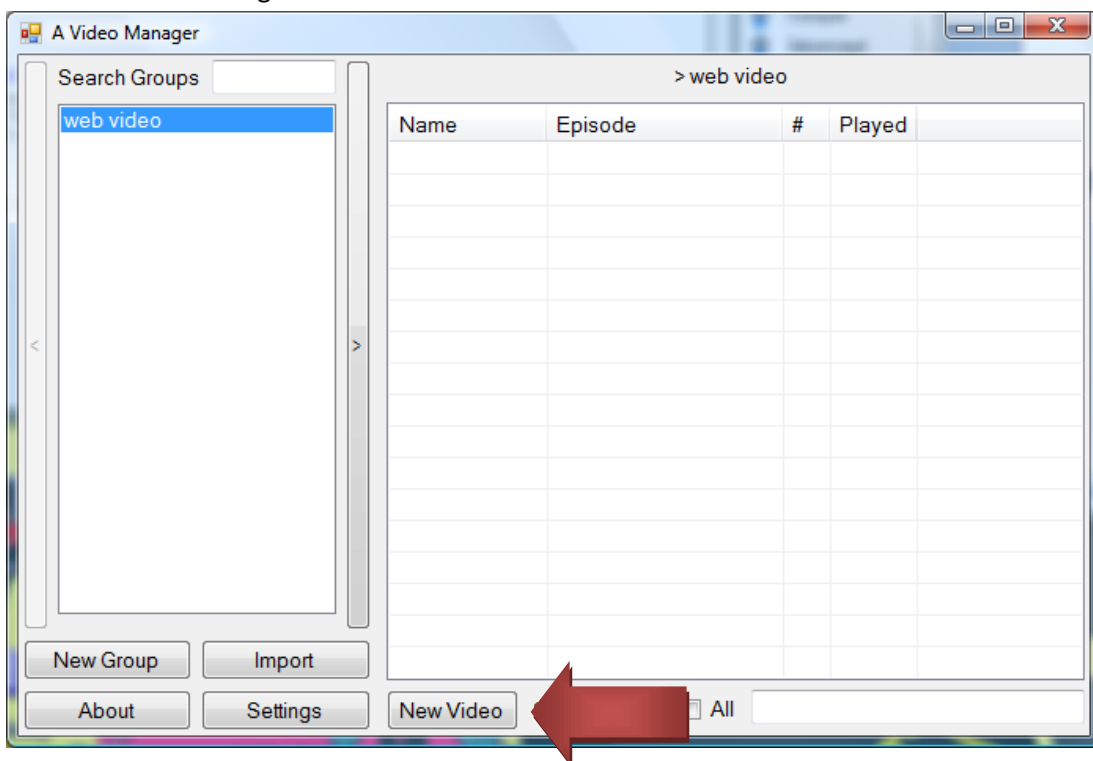
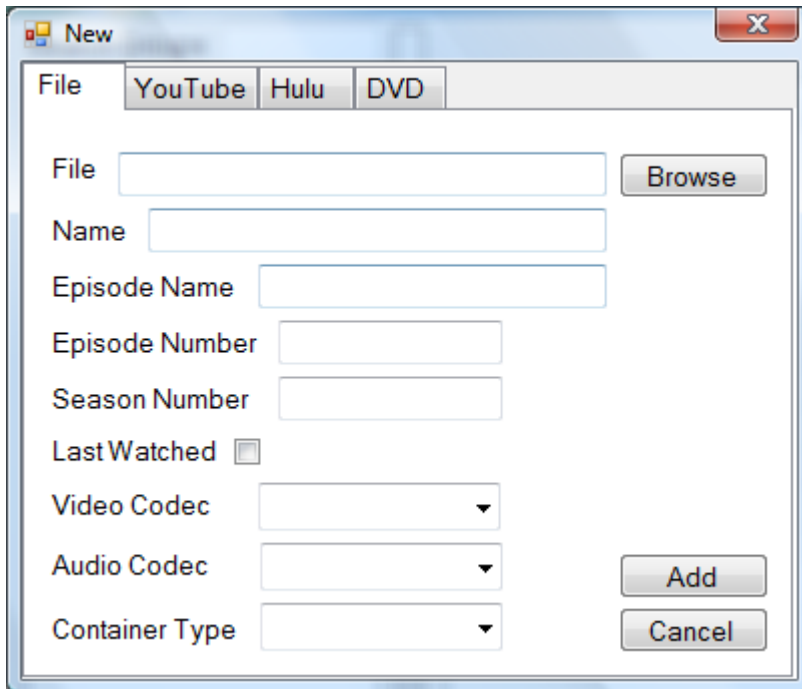


Figure 3

You should now have a window that looks like the pop up in Figure 4.

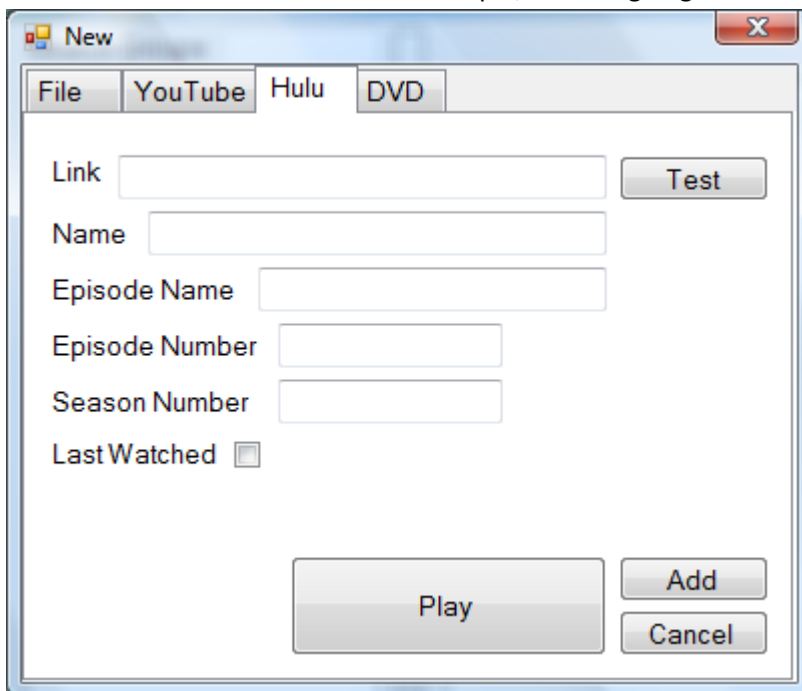


The 'New' dialog box features four tabs: File, YouTube, Hulu, and DVD. The 'File' tab is currently selected. It contains the following fields and controls:

- File:** A text input field with a 'Browse' button to its right.
- Name:** A text input field.
- Episode Name:** A text input field.
- Episode Number:** A text input field.
- Season Number:** A text input field.
- Last Watched:** A checkbox.
- Video Codec:** A dropdown menu.
- Audio Codec:** A dropdown menu.
- Container Type:** A dropdown menu.
- Buttons:** 'Add' and 'Cancel' buttons are located at the bottom right.

Figure 4

To add a YouTube video, select the YouTube tab. If you want to add a Hulu video select the Hulu tab. Both interfaces work the same. For this example, I will be going to the Hulu tab as shown in Figure 5.

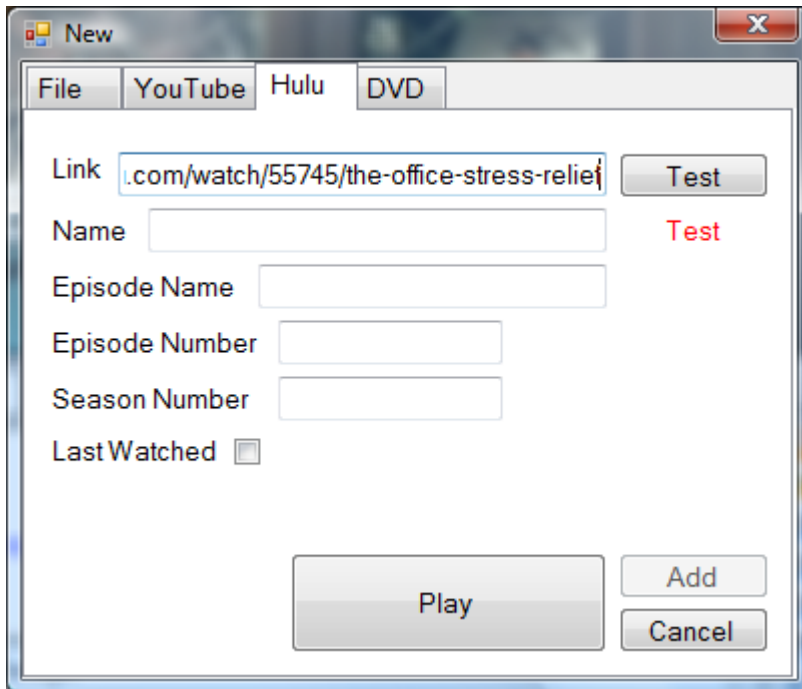


The 'New' dialog box is shown with the 'Hulu' tab selected. It contains the following fields and controls:

- Link:** A text input field with a 'Test' button to its right.
- Name:** A text input field.
- Episode Name:** A text input field.
- Episode Number:** A text input field.
- Season Number:** A text input field.
- Last Watched:** A checkbox.
- Buttons:** 'Play', 'Add', and 'Cancel' buttons are located at the bottom.

Figure 5

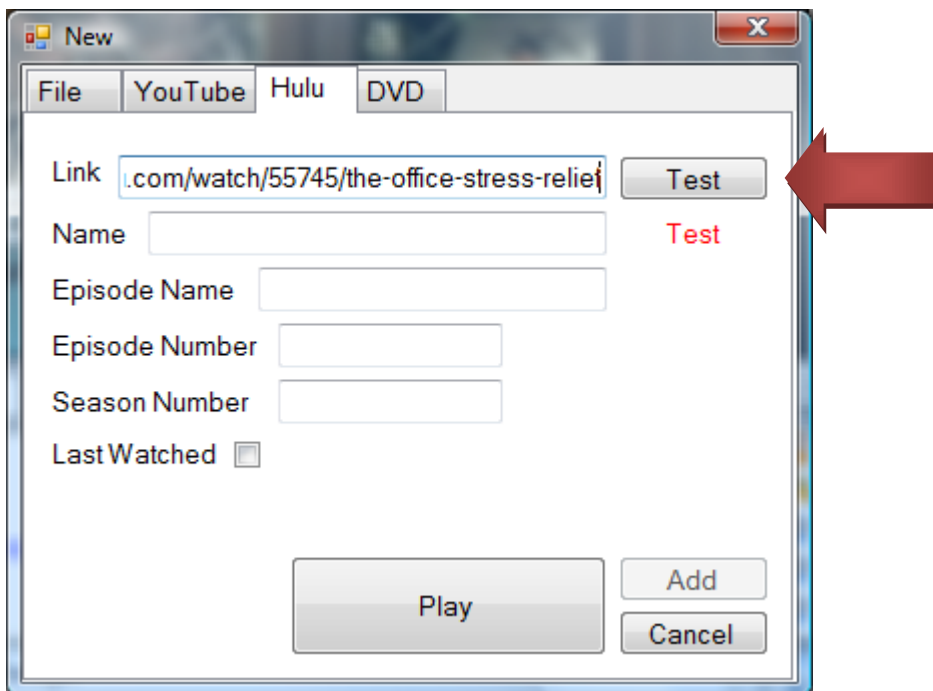
In the text box labeled “Link” type the URL for the video that you want to add. (See Fig 6).



The screenshot shows a 'New' dialog box with tabs for 'File', 'YouTube', 'Hulu', and 'DVD'. The 'YouTube' tab is active. The 'Link' text box contains the URL 'http://www.youtube.com/watch/55745/the-office-stress-relief'. To the right of the 'Link' box is a 'Test' button. Below the 'Link' box are text boxes for 'Name', 'Episode Name', 'Episode Number', and 'Season Number'. To the right of the 'Name' box is a red 'Test' label. At the bottom left is a 'Last Watched' checkbox. At the bottom right are three buttons: 'Play', 'Add', and 'Cancel'.

Figure 6

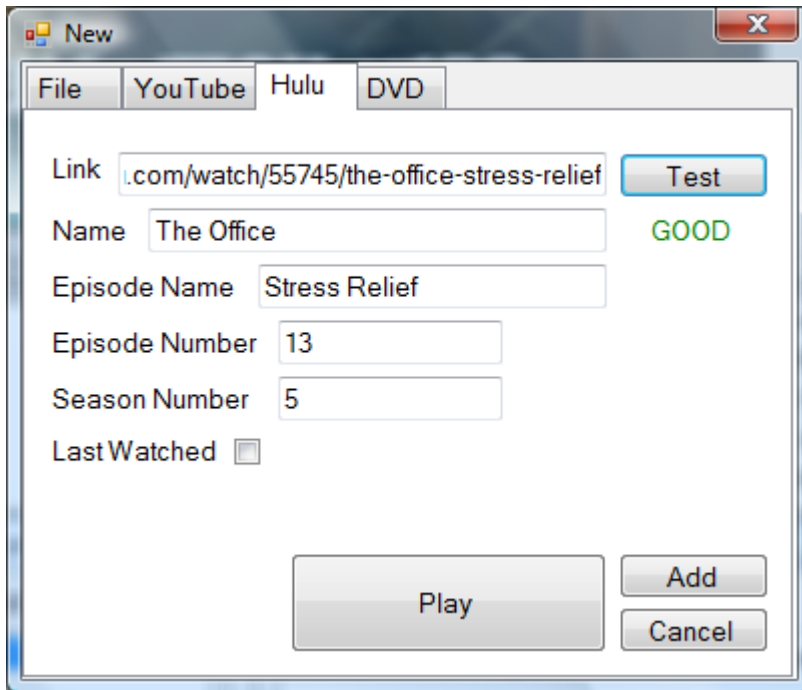
At this point, you must test the link to make sure that it is a valid link. This will also fill out all the data for Hulu videos and the Title for YouTube videos. So click the **Test** button as shown in Figure 7.



This screenshot is identical to Figure 6, showing the 'New' dialog box with the 'YouTube' tab selected and the same URL in the 'Link' field. A large red arrow points from the right side of the dialog box towards the 'Test' button next to the 'Link' field, indicating the next step in the process.

Figure 7

If the link is good, the word “GOOD” will appear under the **Test** button (See Fig 8).

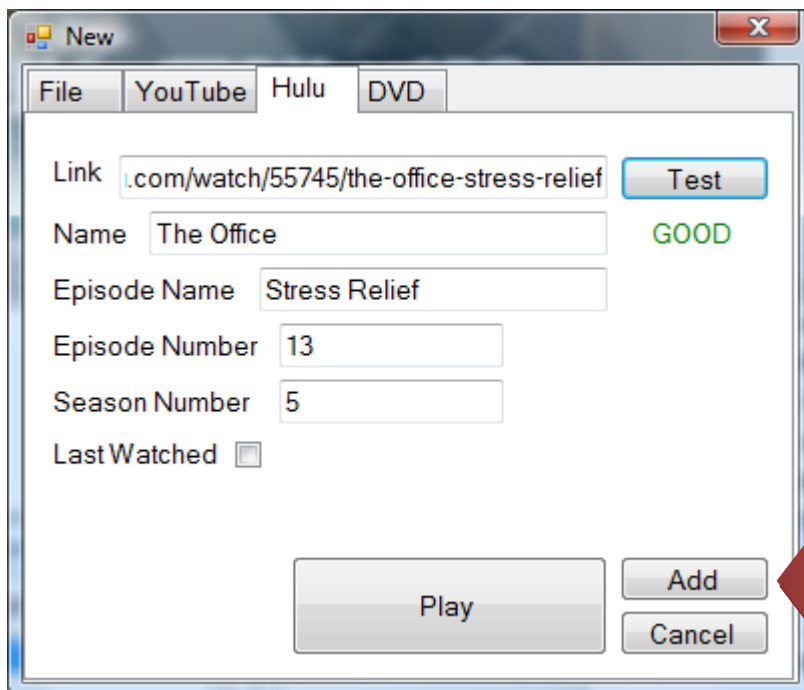


The screenshot shows a window titled "New" with a close button (X) in the top right corner. Below the title bar are four tabs: "File", "YouTube", "Hulu", and "DVD". The "YouTube" tab is selected. The form contains the following fields and buttons:

- Link:** A text box containing ".com/watch/55745/the-office-stress-relief". To its right is a "Test" button.
- Name:** A text box containing "The Office". To its right, the word "GOOD" is displayed in green.
- Episode Name:** A text box containing "Stress Relief".
- Episode Number:** A text box containing "13".
- Season Number:** A text box containing "5".
- Last Watched:** A checkbox that is currently unchecked.
- Buttons:** At the bottom, there is a large "Play" button on the left, and two smaller buttons, "Add" and "Cancel", on the right.

Figure 8

At this point, you can fill in or change any of the data. If you change the “Link”, you will need to test it again. When the data is as you would like it, you can click **Add** as shown in Figure 9. You can always come back and change the information later.



This screenshot is identical to Figure 8, showing the "New" dialog box with the same fields and values. A large red arrow points from the right side of the image towards the "Add" button, which is located at the bottom right of the dialog box, next to the "Cancel" button.

Figure 9

After clicking the **Add** button you will be back at the main window which, if you are using the same link, will look like Figure 10. If you used a different link your screen may look different. It should at least show something now in the list on the right.

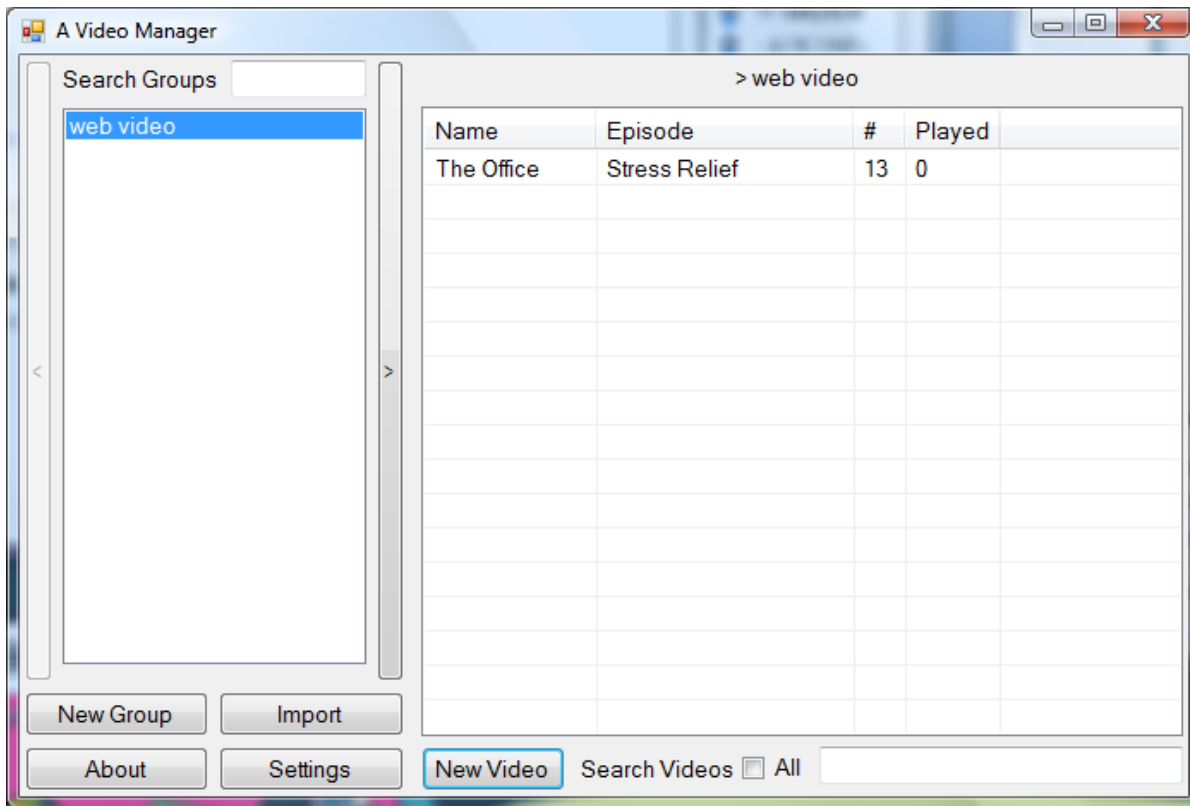


Figure 10

In order to play the video directly from this screen, you right click on “The Office” and a menu drops down. From this menu you can select play. Alternatively, instead of doing a right-click you can simply double-click on “The Office”. When the Viewer appears click **Play** as shown in Figure 11.

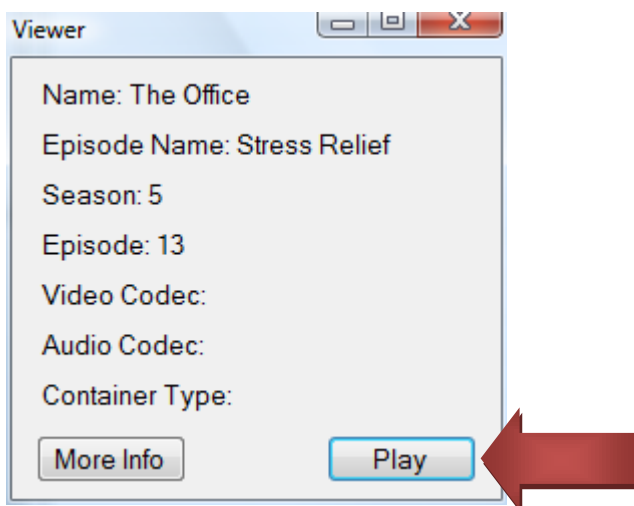


Figure 11

Section 2: Advanced

2.1: The Main Window

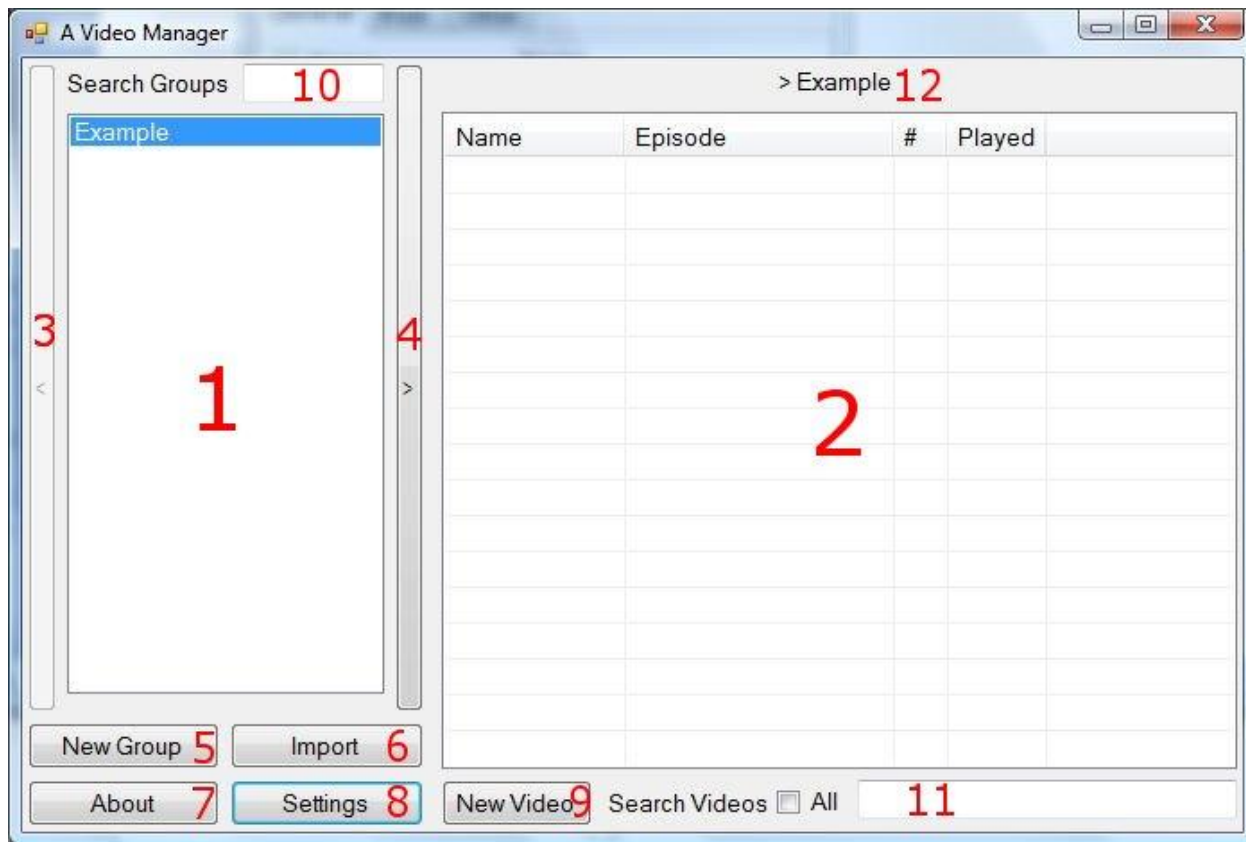


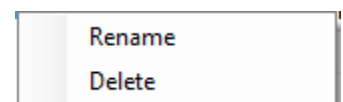
Figure 12

For information on the Main Window look for the subsection with a number matching the numbers in red in Figure 12. For example, the region numbered 2 in Figure 12 is described in section 2.1.2: Video List.

2.1.1: Group List

This section of the main window is called the Group List. It can be thought of as a folder organizer. Each group in this list is like a folder. When a group is selected from the list, as shown in Figure 12, the videos in that group are displayed in section 2. Like folders, groups can also store other groups. In order to view the groups inside of a selected group, you can either double-click on the group name or click on the forward(>) button (4). This will bring you into that group in the Group List. You can keep track of which group you are currently in by viewing the breadcrumbs (12). If you right-click on a group in the Group List you can select from the following options in the drop down menu:

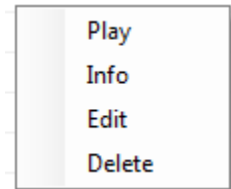
- Rename – Opens a dialog that allows you to rename the selected group.
- Delete – Deletes the currently selected group.



2.1.2: Video List

This section of the main window is called the video list. It displays for the user all of the videos in the currently selected group. By default, if you double-click a video you will see the information you have stored about the selected video. In order to add another video to the list you can just click the **New Video** button (labeled 9) which will bring up the window to add a new video to the list. The right-click menu for items in the Video List include the following:

- Play – Plays the currently selected video.
- Info – Shows you the information about the selected video.
- Edit – Opens a dialog in order to edit the information about the video.
- Delete – Prompts the user if the user wants to delete the selected video.
(This can be turned off in Settings)



2.1.3: Back Button (<)

The Back(<) Button simply allows you to go back one group. This will bring you to the current group's parent group.

2.1.4: Forward Button (>)

The Forward(>) Button allows you to navigate within the the currently selected group. This will be reflected in the breadcrumbs (labeled 12).

2.1.5: New Group Button

The **New Group** button prompts you for a name and then adds a new group with the chosen name to the group list (the new group will be added to the group you are currently in).

2.1.6: Import Button

The **Import** Button will bring up the import dialog (see section 2.3).

2.1.7: About Button

The **About** button brings up information about AVM.

2.1.8: Settings Button

The **Settings** Button will bring up the Settings dialog box (see section 2.2).

2.1.9: New Video Button

The **New Video** Button will bring up the Video Editor window in order to add another video to the currently selected group (see section 2.4).

2.1.10: Search Groups

The Search Groups section allows you to search through all of the groups you have. As you type the auto-search functionality begins the search for you. For example, if you are typing the word "video", when the "v" is typed every group with a v will be displayed. That means you may not need to type the entire word to get the group you are looking for. This is a fast way to find a group if you know its name. Clearing the search box will take you back to the current group.

2.1.11: Video Search

The video search works just like the group search except it is limited to videos in the currently selected group. If you want to search all of the currently stored videos, you just need to check the “All” check box and it will search through all videos.

2.1.12: Breadcrumbs

The breadcrumbs will show you where you are currently located in the group hierarchy.

2.2: Settings Window

The Settings Window is the place where you can customize many attributes of AVM. You can also backup the video information you have stored in AVM. There are three tabs in the Settings Window (see Figure 13). The General Tab allows you to control the columns shown in the video list (2 in Figure 12). The Web Tab allows you to control the way web content works with AVM. Finally, the Other Tab allows you to change many other settings for the program and reset the settings to their default values.

The **Apply** and **Cancel** buttons work as you would expect. **Apply** changes the programs settings and closes the window whereas the **Cancel** button will close the window without applying the new settings. The **Backup** button opens a dialog and allows you to select where you want to save the backup file. This file will have the .xml extension (not all .xml files are AVM backups. It is a common file extension and it is recommended that you select a naming convention so you know it is an AVM backup). This .xml file can be loaded at any time using the Importer (see section 2.3). It is advised that you make backups regularly so that if you have a large number of videos being tracked and you lose a hard drive, you don't lose all your videos. Make sure to store you backup on a device which is external to one which AVM runs on. A flash drive or external hard drive is recommended for storing the backup.

2.2.1: General Tab

The General Tab (Figure 13) is used to control the columns for the video list (2 in Figure 12). The check boxes to the left make the enable the column in the video list. The adjacent textboxes allow you to set the name of the column. For example, by default the “Episode Name” column is just called “Episode”. The size of the columns can be changed at any time at the main window by just dragging them.

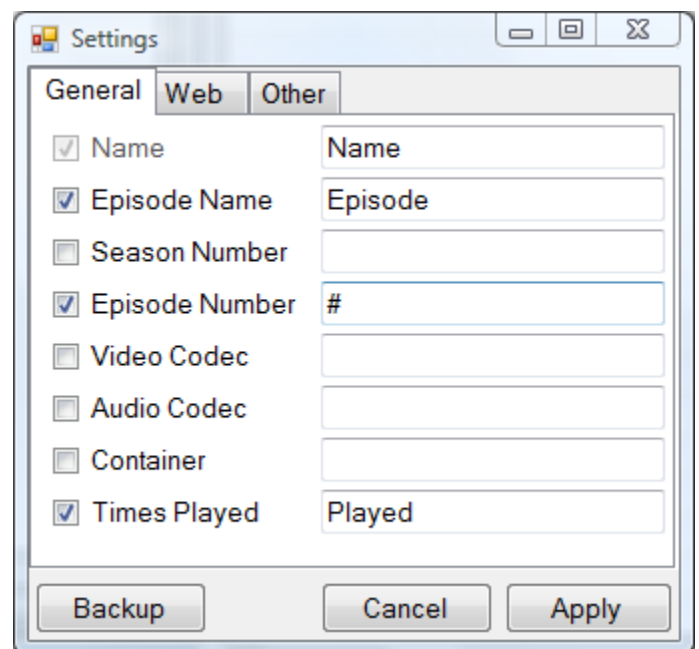


Figure 13

2.2.2: Web Tab

The Web Tab (Figure 14) allows you to control how the web based videos are played. It allows you to watch the video in the web player built into AVM or, if you uncheck the box, you can watch it in your default web browser.

2.2.3: Other Tab

The Other Tab (Figure 15) can potentially be confusing. The first option allows you to reset the settings to their defaults. To do this, press the default button (WARNING: this automatically reverts all settings to their defaults and these are saved).

The next option, the Global File Pattern, specifies the pattern which will be matched to a filename by the file tab of the Video Editor. If, for example, you have a file named "The Office - 1" and your pattern was "Name - EpisodeNumber" then "The Office" would be set as the name and 1 would be set as the episode number.

The next section is the More Info Service. Here you select where you want to get the data from when you press the **More Info** button (in the Viewer). Currently the default choice is imdb, a database of TV shows and movies. Also there is Anime News Network which searches different anime shows. This was added because anime fans were a major group of those interested in the early development of AVM. The last choice is Wikipedia which should be able to give you information on anything the first two cannot find.

The next choice is whether or not you want to be prompted when you delete a video. If you check the box a popup will appear whenever you delete a video.

The next option is the Play on Double-Click option. If this option is checked then when you double-click on a video in the Video List in the main window, the video will play instead of going into the Information Viewer for the selected video.

The next option is "Delete Video when Backspace pressed." If this is checked then when you select a video from the Video List and press the backspace key the selected video will be deleted from AVM.

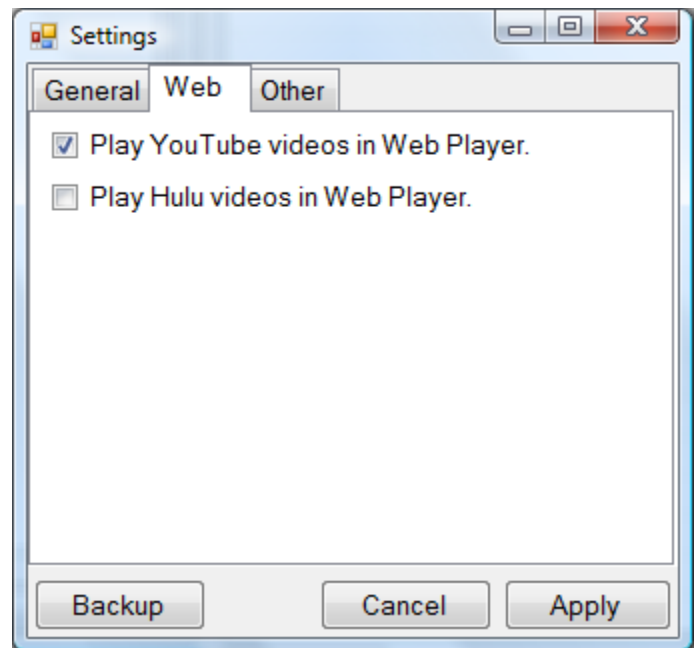


Figure 14

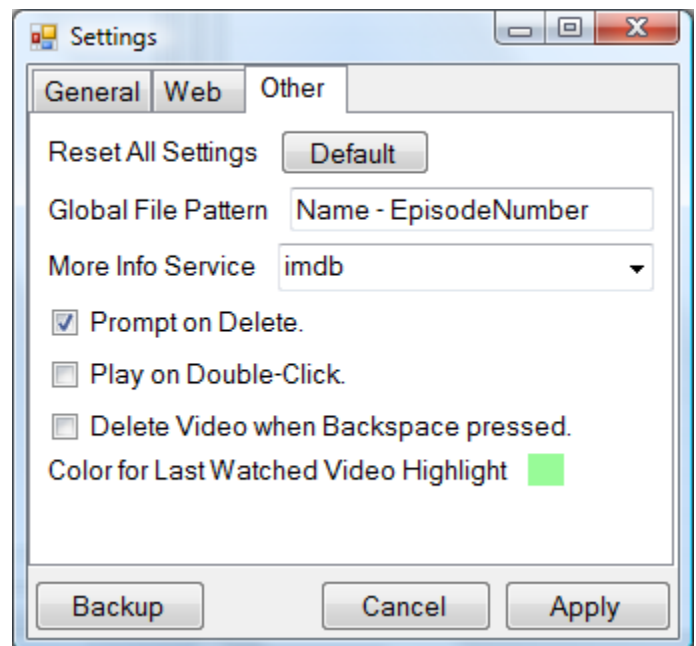


Figure 15

The last option allows you to change the color of the for the last watched video in a series. A series is a group of videos with the same name and generally have episode numbers (can also have other episode information but it is not necessary). If you click on the box then a color chooser dialog will popup allowing you choose a color.

2.3: Importer Window

You get to the Importer by clicking **Import** at the main window (6 in Figure 12). The Importer serves two functions. First, the Importer allows you to import a complete folder filled with videos and other folders into AVM. Second, the Importer allows you to restore backups from the backup files you can create in the Settings (see section 2.2).

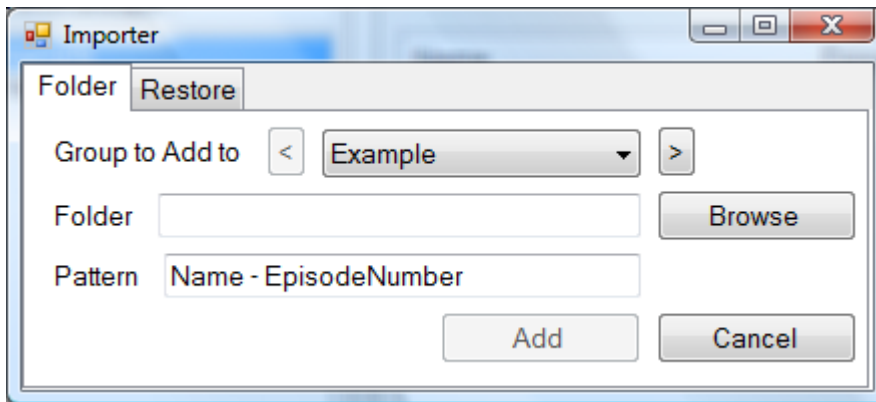


Figure 16

When the importer window appears it will look like Figure 16. The Folder tab allows you to add a folder's files and its sub-folders to AVM. The "Group to Add to" area is for selecting the group which you want to add the folder's content to. The folder hierarchy from the folder will remain with each folder inside the folder becoming a group within the new group. This is very useful if you have a pre-sorted video folder that you keep well organized as it allows you to keep the folder hierarchy.

The next section allows you to select the folder you want to add. To easily select a folder press the **Browse** button and find the folder in the explorer window that will pop up.

The final section is for if you want to adjust the Pattern. This is the pattern that is to be matched for file names. It is, by default, the same as the one set in Settings (Global File Pattern in Figure 15) but can be changed here if you have a folder that has its file's names specially formatted. When you are sure of all this you can press the **Add** button to add the folder to AVM or **Cancel** if you want to back out without adding it.

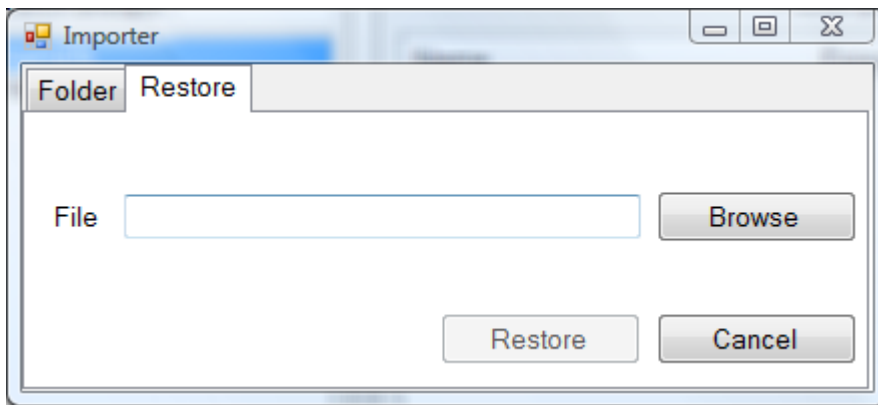


Figure 17

The Restore tab allows you to restore an xml backup that you have made previously. By pressing the **Browse** you can select the desired xml file backup. Next, press **Restore** to restore it. **WARNING: DOING THIS OVERWRITES YOUR CURRENT DATABASE!** This should only be used for backup recovery.

2.4: New Video Window

You can get to the New Video Window by simply hitting the **New Video** button (9 in Figure 12) in the main window when you have a group selected. This window allows you to easily add a new video to the selected group.

When you first open the New Video window the File tab is active. This tab is used if you want to store a video file. Click the **Browse** button and navigate to the file you want to add and select it. AVM will then try and parse information from the filename using the Global File Pattern (Global File Pattern in Figure 15). At this point you can add whatever additional information you know about the video. When you are done, click **Add** to add the video to the collection or click **Cancel** to leave without saving the video.

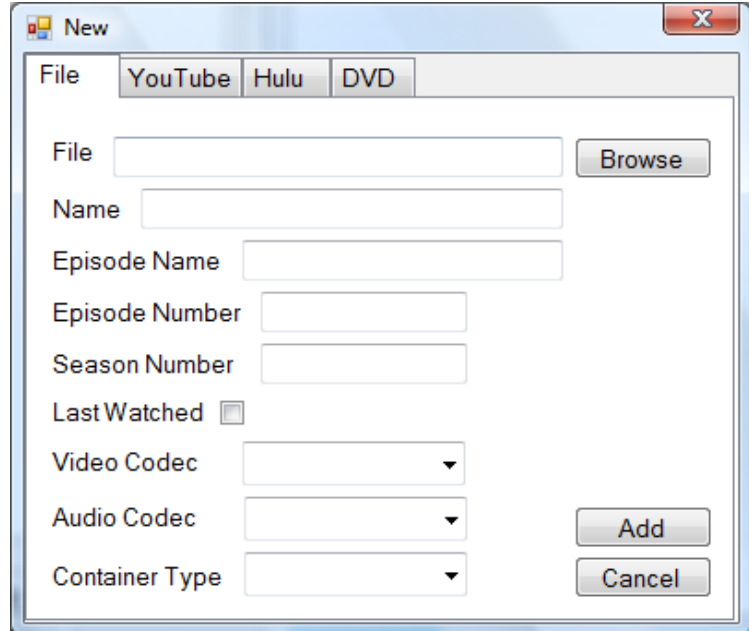


Figure 18

Figure 19 shows the 'New' dialog box with the 'YouTube' tab selected. The dialog has a title bar with a close button. Below the title bar are tabs for 'File', 'YouTube', 'Hulu', and 'DVD'. The 'YouTube' tab is active. It contains the following fields and controls:

- Link:** A text input field followed by a 'Test' button.
- Name:** A text input field.
- Episode Name:** A text input field.
- Episode Number:** A text input field.
- Season Number:** A text input field.
- Last Watched:** A checkbox.
- Buttons:** 'Play', 'Add', and 'Cancel' buttons are located at the bottom right.

Figure 19

Figure 20 shows the 'New' dialog box with the 'Hulu' tab selected. The layout is identical to Figure 19, with the 'Hulu' tab active.

Figure 20

The YouTube and Hulu tabs both work the same way. First, you paste the URL for the video you want to add into the Link box. After you do this, a red “**TEST**” will show up under the **Test** button. This means that you need to test the link before you can proceed. If you click the **Test** button and wait a few seconds, AVM will check to make sure you are linking to a working video and it will populate what information it can about the video from the site. If you want to be sure you have the right URL you can click the **Play** button to see if the correct video plays. When you are finished adding information press **Add** and the video will be added. If you change your mind and do not want to add the video press the **Cancel** button at any time.

REMINDER: When you add web content it is not downloaded to your computer. You will need an Internet connection in order to watch this content.

The DVD tab (Figure 21) can be used to store information about the DVD’s you have in your collection or types of media that you may want to store information for. It is recommended that if you have DVDs to store, the Comments be used to indicate the location of the DVD in your residence. Also you should create a name that will be easy to search. For example, if you have Firefly disc 1 and disc 3 you can make two videos, one being “Firefly disc 1” and the second “Firefly disc 3” or just a single video named “Firefly.” This is up to you, just remember the more videos you store the more you will have to search.

Figure 21 shows the 'New' dialog box with the 'DVD' tab selected. The layout is different from the previous tabs:

- Name:** A text input field.
- Comments:** A large text area for notes.
- Buttons:** 'Add' and 'Cancel' buttons are located at the bottom right.

Figure 21

A Video Manager

Developer Documentation



v1.0

Section 1: About AVM Developer Documentation

This is the main developer documentation for AVM (A Video Manager). This document includes a breakdown of all the different classes in AVM as well as how each of them works. I will do my best so that when you have finished reading this Document you should be able to make fixes and modifications to AVM very easily.

Section 2: What You Need to Know

2.1: How to Setup the Development Environment

2.1.1: Windows

1. Install Visual Studio 2008 (Visual C# 2008 Express Edition <http://www.microsoft.com/express/vcsharp/> will work).
2. Install System.Data.SQLite version 1.0.60.0 from <http://sqlite.phxsoftware.com/>.

2.1.2: Linux

1. Install MonoDevelop for your distribution.
2. Make sure you have the latest version of sqlite3 installed for your distribution (this should be the -dev version if you are using Debian or Ubuntu)
3. If you are grabbing the SVN version make sure to get the "Managed-Only" version of System.Data.SQLite from <http://sqlite.phxsoftware.com/>.

2.2: Terms

Within AVM there are a few terms that you must understand before going further.

- Node – In AVM a node is the object which stores all of the information about a video file.
- Group – In AVM a group is like a folder. It can store other groups as well as nodes.

2.3: The Database

This is a list of the Database tables used in AVM. In nodes the elements match up with properties in the Node, EpisodeInfo, and FileData classes. In groups the elements match up with the properties for the Group class. All calls to the database should be in the Database class. This is to contain the database calls to one spot in order to keep better track of them. Also each public function should Open AND Close the connection.

2.3.1: nodes

- node_id – INTEGER primary key & auto increments
- name – TEXT
- watched – INTEGER defaults to 0
- type – INTEGER
- url – TEXT
- embedded – TEXT

- comment – TEXT
- episode_number – INTEGER
- season_number – INTEGER
- last_watched – INTEGER
- episode_name – TEXT
- uri – TEXT
- video_encoding – TEXT
- audio_encoding – TEXT
- container – TEXT
- parent_group_id – INTEGER

2.3.2: groups

- group_id – INTEGER primary key & auto increments
- name – TEXT
- parent_id – INTEGER

2.4: Adding New Features

2.4.1: Parsers

If you want to add an additional parser you should create a new class within the Parsers folder also make sure that it is setup within the AVM.Parsers namespace. Next you want to make sure that all the information is provided to the constructor that is to be used in the parsing. The Parse() function should then be created that will do most of the work though it should not return any data. Data should be stored in properties with names that match the name they will take in the AVM.Types.Node object.

2.4.2: Adding New Types of Videos

If you want to add another video site to the ones currently in AVM this is not hard to do for web based formats. If you have a new site in mind, first add a tab to the NodeEditor for the new site. Then you must create the fields for it. I personally recommend copying the YouTube or Hulu tabs completely and reuse the code by just changing some of the calls. These calls are the next additions. First, you must add a new parser. To see how to do this best, read section 2.3.1. Next you must create a new property within the AVM.Types.Node class to read in the new website. This is used to set the type number in the database. The url type also must be set in order for the new site to work.

2.4.3: Adding Additional Services to “More Info”

In order to add an additional services to the “More Info” button there are two places which need to be changed. First, in the Settings form an item must be added in order to select the service. Second, the same name must be added to the case statement in the NodeViewer’s `moreInfoButton_Click` function. Inside this case statement you merely add the needed case and add what is to be done by the service within the case. No additional changes are needed. There are three options currently and they can be used as examples for how to add a new search service for the “More Info” button.

2.4.4: Adding Additional Types to File Parsing

The types available in file parsing may seem limiting. Adding additional types to the file parser is not hard. All that must be added is an if statement to the `parseFile` function in the `FolderParser`. Use the currently available types as examples of how to layout the if statement. `Name` can be used as an example of how to parse a type that will create a string while `EpisodeNumber` is a good example of how to parse a type that will create an integer.

2.5: Important Class Information

2.5.1: The AVM Namespace

The AVM namespace is laid out in a way so that it is easy to work with adding parsers and more unique types for AVM to use. The first thing to notice is every class for AVM is within the AVM namespaces. Within this there are two sub classes. These are `Types` which are storage classes (including `Group`, `Node`, `FileInfo`, and `ExpisodeInfo`) and then there are `Parsers` which are any parsers that are created to be used within AVM. Parsers include all forms of parsers meaning both parsers for web pages and for grabbing file information should both be within the `AVM.Parsers` namespace.

2.5.2: The Node Class

The `Node` class is one of the most central Classes to AVM. The `Node` class stores all the data on a specified node or video. It is made up of 3 parts. The first is the `Node` class itself which holds all the general information on a video as well as information for web videos. Next there is the `Episode` section of the `Node` class. This stores all the information relating the video being an episode. Information like `Episode Name` and `Episode Number` as well as if it was the `Last Watched` in a series. The last section of the `Node` class is the `File` section. This holds onto information about the file specifically. This is information like the filename and path as well as information about the codec's and containers used by the file. Both the `Episode` and `File` information are stored within a `Node`. They are separate classes though allowing them to not take up memory when the `Node` is loaded if less common information (`File` or `Episode` related) is not provided for the `Node`.

Section 3: The Classes

3.1: About

This section will have a breakdown of every class within the AVM namespace by what part of the namespace it is in (`AVM`, `AVM.Parsers`, or `AVM.Types`).

3.2: AVM

3.2.1: AboutBox

`AboutBox` is a class which displays the standard .net `About` window showing the basic information about AVM. This class has no real modifications from the default `AboutBox` provided by Visual Studio .NET 2008.

3.2.2: Database

The Database class is used for interfacing with SQLite database which stores all the data for AVM. There should be no SQL calls used outside of this class. The database is stored in the users AppData folder under local/AVM and is called "Video Manager.db3".

Properties

- `public long ParentGroup` – This is used to set or get the parent of the CurrentGroup.
- `public long CurrentGroup` – This is used to set or get the Current group being used by the SQLite database. If it is set it will automatically change the ParentGroup.

Constructors

- `public Database(string databasePath)` – This creates a Database at the path sent via databasePath. If it doesn't exist then it will create it.

Methods

- `public void refreshGroups(System.Windows.Forms.ListBox listBox)` – Fills listBox with the group that is currently selected.
- `public List<AVM.Types.Group> getGroups(int groupId)` – Returns a List of all the groups inside the specified group.
- `public List<AVM.Types.Group> getAllGroups()` – Returns a List of all the groups in the database.
- `public void refreshComboBoxGroups(System.Windows.Forms.ComboBox comboBox)` – This will clear and populate a ComboBox with the Groups that are in the current ParentGroup.
- `public void searchGroups(System.Windows.Forms.ListBox listBox, string query)` – Fills listBox with the groups found using the passed in query string.
- `public void fillGroups(List<AVM.Types.Group> groups)` – Takes a List of groups and fills the database with them.
- `public void addGroup(string name)` – This will add a group using the name provided as its name and the current group as its parent.
- `public void renameGroup(AVM.Types.Group group, string newName)` – Renames the group passed in from the database using the string that is passed in.
- `private void recursiveRemoveGroup(AVM.Types.Group group, SQLiteTransaction trans)` – This function will recursively delete a whole group, its nodes and sub-groups. NOTE: This requires an open transaction and needs to be committed or rolled back after.
- `public void removeGroup(AVM.Types.Group group)` – Remove the group sent to the function from the database.
- `public long getLastGroupId()` – Returns the last group id which is also the largest.
- `public void gotoParent()` – This moves the current group back to its parent.
- `public List<AVM.Types.Node> getAllNodes()` – This function returns a List of all the nodes in the database.
- `private List<AVM.Types.Node> selectNodesInGroup(long groupId, SQLiteTransaction trans)` – This will return a List of Nodes that are in the group specified by groupId. NOTE: This requires an open transaction and needs to be committed or rolled back after.

- `private void removeNodesFromGroup(long groupId, SQLiteTransaction trans)` – This will remove all Nodes from the groupId sent in. NOTE: This requires an open transaction and needs to be committed or rolled back after.
- `public void refreshNodes(ref System.Collections.Generic.List<AVM.Types.Node> list, ref List<AVM.Types.Group> groups)` – Fill the List passed with all the nodes in the current group.
- `public void fillNodes(List<AVM.Types.Node> nodes)` – Adds all the nodes in the supplied List to the database.
- `public void addNode(AVM.Types.Node node)` – Add the supplied node to the database.
- `public void updateNode(AVM.Types.Node old_node, AVM.Types.Node node)` – Update the supplied node with the new nodes information.
- `private void removeSingleNode(AVM.Types.Node node, SQLiteTransaction trans)` – Removes a single node from the database. NOTE: A connection must be open and pass in a transaction and after this is run it must be committed or rolled back.
- `public void removeNode(AVM.Types.Node node)` – Remove the node that is supplied.
- `public void incrementPlayed(AVM.Types.Node node)` – Increment the times played for a node.
- `public void searchNodes(ref List<AVM.Types.Node> nodes, string query, bool searchAll)` – Search the database using the query and return it in the List. NOTE: This can search only the group or in all groups.
- `public long getLastNodeId()` – Gets the last node id which is also the largest node id.
- `public void lastWatched(AVM.Types.Node node)` – Sets the node passed in as the last watched.
- `public void Clear()` – This function empties the database.
- `public string findBreadcrumbs(long starting_id)` – This returns a breadcrumb list of all the groups going back to the root group from the group whose id is supplied.
- `public void Kill()` – This closes the database.

3.2.3: DeleteConfirmation

Asks the user for confirmation before a video is deleted from the database.

Properties

- `public bool Delete` – Returns true if the user wants to delete the selected video.

Constructors

- `public DeleteConfirmation()` – Creates the DeleteConfirmation form.

Methods

- `private void deleteButton_Click(object sender, EventArgs e)` – This is called when the "Delete" button is pressed. It sets `_delete` to true and then closes the dialog.
- `private void cancelButton_Click(object sender, EventArgs e)` – This is called when the "Cancel" button is pressed. It sets `_delete` to false and then closes the dialog.

3.2.4: GroupEditor

Allows the user to edit the name of the selected group.

Properties

- `public string NewName` – Returns the new name for the group. Returns "" if no new name is created.

Constructors

- `public GroupEditor(string originalName, string formName, string buttonName)` – Creates a GroupEditorDialog using the originalName passed in as the base name. Also the title can be configured using formName and the button can be titled using buttonName so that the Editor can be used both for addition and editing of groups.

Methods

- `private void okButton_Click(object sender, EventArgs e)` – Closes the dialog so that the MainForm can get the NewName.
- `private void cancelButton_Click(object sender, EventArgs e)` – Closes the form and makes it so NewName will return "".
- `private void renameTextBox_TextChanged(object sender, EventArgs e)` – Saves the text in the TextBox to _name every time it's changed.
- `private void renameTextBox_KeyPress(object sender, KeyPressEventArgs e)` – Allows the user to press "enter" in order to finish using this dialog.

3.2.5: Importer

Imports groups and videos from either a backup xml document or from a specified folder.

Properties

- `public bool RestoreTookPlace` – Tells if a restore has taken place so the main window can react.

Constructors

- `public Importer(MainForm avm, Database database, int selectedGroup)` – Creates an Importer form passing the MainForm, database and currently selected group.

Methods

- `private void Importer_Load(object sender, EventArgs e)` – Runs on load. Populates comboBox on the folder tab and populates the patternTextBox.
- `private void groupForwardButton_Click(object sender, EventArgs e)` – Moves the comboBox forward in the groups list based off the currently selected group in the ComboBox.
- `private void groupBackButton_Click(object sender, EventArgs e)` – Moves the comboBox back in the groups list.
- `private void backupBrowseButton_Click(object sender, EventArgs e)` – Opens up the openFileDialog so that the backup xml file can be selected to restore data from.
- `private void topTextBox_TextChanged(object sender, EventArgs e)` – Enables and disables the backupAddButton based on if there is currently a file selected.

- `private void backupAddButton_Click(object sender, EventArgs e)` – Runs the backup xml parser and restores the database from it. Also refreshes the group list.
- `private void backupCancelButton_Click(object sender, EventArgs e)` – Closes the form.
- `private void folderBrowseButton_Click(object sender, EventArgs e)` – Displays the folderBrowserDialog and selects the folder to be added.
- `private void folderAddButton_Click(object sender, EventArgs e)` – Runs the FolderParser based on the selected folder and refreshes the group list.
- `private void folderFileTextBox_TextChanged(object sender, EventArgs e)` – Enables and disables the folderAddButton based on if there is currently a folder selected.
- `private void folderCancelButton_Click(object sender, EventArgs e)` – Closes the form.

3.2.6: MainForm

This is the main form for the program.

Constructors

- `public MainForm()` – Basic class constructor. Literally nothing beyond InitializeComponets takes place here.

Methods

- `private void newGroupButton_Click(object sender, EventArgs e)` – Prompts the user for a group name and if it gets a name it adds it to database and then refreshes the groupListBox.
- `private void settingsButton_Click(object sender, EventArgs e)` – Displays the Settings dialog for the user. If the columns are changed then it reformats the columns and reloads the nodes.
- `private void aboutButton_Click(object sender, EventArgs e)` – Displays the About dialog for the user.
- `private void importButton_Click(object sender, EventArgs e)` – Displays the Importer dialog for the user which can restore backed up databases and can also cause both group and node refreshes.
- `private void newButton_Click(object sender, EventArgs e)` – Creates a new node by running the NodeEditor in "New" mode. Also refreshes nodes.
- `private void nodeSearchAllCheckBox_CheckedChanged(object sender, EventArgs e)` – If there is a current search on the nodes then it will redo the search based off of the new status of if it should be searching all nodes or just nodes in the current group. May refresh nodes.
- `private void nodeSearchTextBox_TextChanged(object sender, EventArgs e)` – If search is entered make sure to change current nodes listed to the search results. Different searches take place based off of the checkbox for searching all nodes. Refreshes nodes.
- `private void groupBoxMenu_Opening(object sender, CancelEventArgs e)` – Enables and disables the options in the group right-click menu based on whether or not a group is currently selected.
- `private void addNewGroupMenuItem_Click(object sender, EventArgs e)` – Creates a new group by running the GroupEditor. Refreshes groups.

- `private void renameMenuItem_Click(object sender, EventArgs e)` – Renames currently selected group using the GroupEditor. Refreshes groups.
- `private void deleteGroupMenuItem_Click(object sender, EventArgs e)` – Deletes currently selected group. If PromptOnDelete is true it will prompt the users using DeleteConfirmation. Refreshes groups.
- `private void nodeBoxMenu_Opening(object sender, CancelEventArgs e)` – Enables and disabled the options in the node right-click menu based on whether or not a node is currently selected.
- `private void playNodeMenuItem_Click(object sender, EventArgs e)` – Runs playSelectedNode.
- `private void infoNodeMenuItem_Click(object sender, EventArgs e)` – Opens the NodeViewer for the currently selected node. Runs play if the viewer wants the currently selected node to be played.
- `private void editNodeMenuItem_Click(object sender, EventArgs e)` – Edits the currently selected node using NodeEditor. Refreshes nodes.
- `private void deleteNodeMenuItem_Click(object sender, EventArgs e)` – Deletes the currently selected node. If PromptOnDelete is selected then DeleteConfirmation will ask for confirmation. Refreshes nodes.
- `private void deleteNodeMenuItem_Click(object sender, EventArgs e)` – Deletes the currently selected node. If PromptOnDelete is selected then DeleteConfirmation will ask for confirmation. Also refreshes nodes.
- `private void switchGroupToSelected(object sender, EventArgs e)` – Switches the current group to the last currently selected one. Traverses into the group that is selected.
- `private void backOneGroup(object sender, EventArgs e)` – Switches the current group to its parent. Traverses backward into the parent group.
- `private void enableForwardBackButtons()` – Makes sure that the "<" and ">" buttons are correctly enabled.
- `private void searchGroupTextBox_TextChanged(object sender, EventArgs e)` – If text changes then it does a search based on what is in the textbox and places the results in the group list. Refreshes groups.
- `private void groupListBox_SelectedIndexChanged(object sender, EventArgs e)` – When a different group is selected refresh the node list based on what is in the newly selected group. Refreshes nodes.
- `public void refreshGroups()` – Refreshes the groups from the database.
- `private void loadNodes()` – Populates the nodeListView based on what is supposed to be shown.
- `private void nodeListView_ColumnWidthChanged(object sender, ColumnWidthChangedEventArgs e)` – Saves a changed ColumnWidth to the settings.
- `private void nodeListView_DoubleClick(object sender, EventArgs e)` – If DoubleClickPlay is enabled then this will just play a video. If not then it will just display a NodeViewer.
- `public void playSelectedNode()` – Plays the selected node. If it is an episode it will set it as lastWatched. Refreshes nodes. (because of possible lastWatched change)
- `private void nodeListView_KeyPress(object sender, KeyPressEventArgs e)` – Checks for the user pressing the backspace key when a node is selected. If PromptOnDelete is enabled will prompt user with DeleteConfirmation otherwise will just delete the video when backspace is pressed and BackspaceDelete is enabled.

- `private void MainForm_Load(object sender, EventArgs e)` – Creates the MainForm and also loads or creates the database. Refreshes both nodes and groups to their last selected.
- `private void MainForm_FormClosing(object sender, FormClosingEventArgs e)` – Saves current settings and positions for group and nodes. Also kills the database.
- `private void formatColumns()` – Sets up all of the columnWidths and columnNames.

3.2.7: NodeEditor

This form is used to gather information on the currently selected node or new node.

Properties

- `public bool Successful` – Returns true if the information in NodeEditor is supposed to be used to populate/update the database. If false does nothing with it.
- `public AVM.Types.Node Node` – Returns the Node with the edited or new data in it.

Constructors

- `public NodeEditor(string title, AVM.Types.Node node)` – Populates the editor with data if a node is provided. Changes the text on buttons based on if the title is "new" or not.

Methods

- `private void fileBrowseButton_Click(object sender, EventArgs e)` – Shows the file selection dialog in order to prompt the user for the file they want to add.
- `private void fileVideoCodecComboBox_SelectedIndexChanged(object sender, EventArgs e)` – Stores the new Video Encoding in a node and propagates it to all tabs.
- `private void fileAudioCodecComboBox_SelectedIndexChanged(object sender, EventArgs e)` – Stores the new Audio Encoding in the new node and propagates it to all tabs.
- `private void fileContainerComboBox_SelectedIndexChanged(object sender, EventArgs e)` – Stores the new Container in the new node and propagates it to all tabs.
- `private void youtubeLinkButton_Click(object sender, EventArgs e)` – Parses the YouTube link in the YouTube textbox. If valid displays GOOD else displays BAD. If valid will enable the Add buttons.
- `private void youtubeLinkTextBox_TextChanged(object sender, EventArgs e)` – Runs when user changes the YouTube link. Changes the test label to Test. Disables the Add buttons.
- `private void huluLinkButton_Click(object sender, EventArgs e)` – Parses the YouTube link in the Hulu textbox. If valid displays GOOD else displays BAD. If valid will enable the Add buttons.
- `private void huluLinkTextBox_TextChanged(object sender, EventArgs e)` – Runs when user changes the Hulu link. Changes the test label to Test. Disables the Add buttons.
- `private void updateNodeName(string name)` – Updates the NodeName and every textbox that is related to it.
- `private void updateEpisodeName(string name)` – Updates the EpisodeName and every textbox that is related to it.
- `private void updateLastWatched(bool value)` – Updates the LastWatched status and every checkbox related to it.
- `private void updateEpisodeNumber(int value)` – Updates the EpisodeNumber and every textbox related to it.

- `private void updateSeasonNumber(int value)` – Updates the SeasonNumber and every textbox related to it.
- `private void episodeOrSeasonNumber_TextChanged(object sender, EventArgs e)` – Updates either the season or the episode depending on which textbox was changed.
- `private void name_TextChanged(object sender, EventArgs e)` – Runs `updateNodeName` when a textbox related to the name is changed.
- `private void episodeName_TextChanged(object sender, EventArgs e)` – Runs `updateEpisodeName` when a textbox related to the name is changed.
- `private void dvdCommentsTextBox_TextChanged(object sender, EventArgs e)` – Updates the comment in the node when the comment textbox was changed.
- `private void lastWatched_CheckedChanged(object sender, EventArgs e)` – Runs the `updateLastWatched` when a checkbox related to it is changed.
- `private void playButton_Click(object sender, EventArgs e)` – Plays the web video to test the link.
- `private void addButton_Click(object sender, EventArgs e)` – Marks as success and closes form.
- `private void cancelButton_Click(object sender, EventArgs e)` – Marks success as false and closes form.

3.2.8: NodeViewer

This form is used to show the user the information on a selected node.

Properties

- `public bool Play` – Tells whether or not the video should be played after the viewer is closed.

Constructors

- `public NodeViewer(AVM.Types.Node node)` – Creates a NodeViewer form populated with the information from the passed in node.

Methods

- `private void playButton_Click(object sender, EventArgs e)` – Sets play to true and closes the form.
- `private void moreInfoButton_Click(object sender, EventArgs e)` – Opens a web browser to a search page for the chosen service.
- `private void NodeViewer_Load(object sender, EventArgs e)` – Loads all of the information stored in `_node`.

3.2.9: Settings

This form is used for the user to change the settings for AVM.

Properties

- `public bool Success` – Returns whether or not to refresh the `nodeListView` columns.

Constructors

- `public Settings(Database db)` – Creates a new Settings form and passes in `db` incase a backup is made.

Methods

- `private void applyButton_Click(object sender, EventArgs e)` – Applies all the changes made in Settings.
- `private void cancelButton_Click(object sender, EventArgs e)` – Closes the form without doing anything else. (Does not save settings)
- `private void backupButton_Click(object sender, EventArgs e)` – Prompts for a file to save the xml backup to and creates an xml backup from the current database.
- `private void resetButton_Click(object sender, EventArgs e)` – Resets settings to their defaults and reloads the form.
- `private void lastWatchedColorPictureBox_Click(object sender, EventArgs e)` – Loads a color select dialog to change the last watched highlight color.
- `private void Settings_Load(object sender, EventArgs e)` – Loads the properties when the form loads.
- `private void loadProperties()` – Loads all the fields properly from the stored settings.

3.2.11: WebPlayer

This form plays YouTube and Hulu content.

Properties

- `public string Title` – Returns the title of the form.

Constructors

- `public WebPlayer()` – Makes a new WebPlayer.

Methods

- `public void PlayYouTube(string embedded)` – Plays a YouTube video using the embedded string.
- `public void PlayHulu(string url)` – Plays a Hulu video by using the embedded url.
- `private void WebPlayer_Resize(object sender, EventArgs e)` – Refresh the YouTube video to the new size when a YouTube video is playing.
- `private void WebPlayer_FormClosing(object sender, FormClosingEventArgs e)` – Prevents errors from showing up when the form is closed.

3.3. AVM:Types

3.3.1: EpisodeInfo

This class stores information about an episode if an episode is stored.

Properties

- `public int EpisodeNumber` – Episode number for the node.
- `public int SeasonNumber` – Season number for the node.
- `public string EpisodeName` – Name of the episode for the node.
- `public bool LastWatched` – Whether or not the episode was the last one watched.

Constructors

- `public EpisodeInfo()` – Creates a basic version of EpisodeInfo.

- `public EpisodeInfo(int episodeNumber)` – Creates EpisodeInfo with a populated episode number.

3.3.2: FileData

Stores information about an episode if a file is stored.

Properties

- `public Uri Uri` – The file's uri (filename).
- `public string Video_Encoding` – Stores and returns valid video encoding names.
- `public string Audio_Encoding` – Stores and returns valid audio encoding names.
- `public string Container` – Stores and returns valid container names.

Constructors

- `public FileData()` – Creates a basic version of FileData.
- `public FileData(string fullPath)` – Creates FileData using the full path to the file.
- `public FileData(string path, string name)` – Creates FileData using the path and name of the file.

3.3.3: Group

Stores the information about a group.

Properties

- `public string Name` – The name of the group.
- `public long Id` – The group_id for the group. Used to identify the group in database lookups.
- `public long ParentId` – The group_id for the parent of the current group. Used to identify parent group in database lookups.

Constructors

- `public Group()` – Creates a basic group with no name.
- `public Group(string name, long id)` – Creates a group with a name and id.
- `public Group(string name, long id, long parentId)` – Creates a group with a name, id and id of its parent group.

Methods

- `public override string ToString()` – Returns the name of the group.

3.3.4: Node

Stores the data on a node as well as EpisodeInfo and FileData if they are needed.

Properties

- `public long Id` – The node_id for the node. Used as an identifier for database lookups.
- `public string Name` – The nodes name that is seen by the user.
- `public bool IsFile` – Checks if there is FileData in this node.
- `public bool IsYouTube` – Checks if the node is a YouTube video.

- `public bool IsHulu` – Checks if the node is a Hulu video.
- `public bool IsEpisode` – Checks if the node has EpisodeInfo.
- `public AVM.Types.EpisodeInfo Episode` – Returns or sets the EpisodeInfo for this node.
- `public AVM.Types.FileData File` – Returns or sets the FileData for this node.
- `public string YouTube` – Used to set a YouTube video by sending the embedded link. If an embedded link is sent then it creates the right html code for an embedded player and if it is null it doesn't.
- `public string Hulu` – Used to set a Hulu video by sending the embedded link. If an embedded link is sent then it creates the right html code for an embedded player (currently not working) and if it is null it doesn't.
- `public short UrlType` – Returns or sets what type of Url is stored in Url.
0: nothing
1: youtube
2: hulu
- `public string Url` – Returns or sets the url for the YouTube or Hulu video.
- `public string embedded` – Gets and sets the embedded string. NOTE: Do not use this to set the embedded string for a new YouTube or Hulu video.
- `public string Comment` – Gets or sets the comment. This if for DVDs.
- `public int TimesPlayed` – Gets and sets the number of times played.
- `public long ParentId` – The group_id of the parent group for the node.

Constructors

- `public Node()` – Creates a basic node with no information and a blank name.
- `public Node(string path)` – Creates a node based on a given path to a file.

Methods

- `public void Play()` – Used to play Hulu and YouTube videos.

3.4. AVM:Parsers

3.4.1: BackupParser

Handles both reading and writing XML based backup files.

Constructors

- `public BackupParser(Database db)` – Creates a basic parser with a database if needed.

Methods

- `public void WriteXmlBackup(string file)` – Writes an xml backup to the file passed in.
- `private void writeNodes(XmlTextWriter xmlWriter)` – Writes all the nodes to the xmlWriter.
- `private void writeNode(AVM.Types.Node node, XmlTextWriter xmlWriter)` – Writes out an individual node to the xmlWriter.

- `private void writeGroups(XmlTextWriter xmlWriter)` – Writes all the groups to the xmlWriter.
- `private void writeGroup(AVM.Types.Group group, XmlTextWriter xmlWriter)` – Writes out an individual group to xmlWriter.
- `public void ReadXmlBackup(string file)` – Reads an xml backup from the file passed in.
- `private void readNodes(XmlTextReader xmlReader)` – Reads all the nodes from the xmlReader and adds them to the database.
- `private AVM.Types.Node readNode(XmlTextReader xmlReader)` – Reads in an individual node from the xmlReader and returns a node.
- `private void readGroups(XmlTextReader xmlReader)` – Reads all the groups from the xmlReader and adds them to the database.
- `private AVM.Types.Group readGroup(XmlTextReader xmlReader)` – Reads in an individual group from the xmlReader and returns a group.

3.4.2: FolderParser

Parses a folder and its sub folders and then stores the hierarchy in the database with the files as nodes.

Properties

- `public string Pattern` – Sets the pattern used to parse the filename.

Constructors

- `public FolderParser()` – Creates a basic FolderParser.
- `public FolderParser(string path, AVM.Database db)` – Creates a Folder parser for a given path and using the passed in database.

Methods

- `private void parseFolder(DirectoryInfo folder, long parent_id)` – Parses a single folder for files and additional folders.
- `public AVM.Types.Node parseFile(long parent_group, FileInfo file)` – Parses a single file using the pattern.
- `public void parse(long parent_id)` – Parses through the path it is configured to parse.
- `private string cleanString(string original)` – Removes everything between a () and [] in the string.

3.4.3: HuluParser

Parses the data from a Hulu page that has a video.

Properties

- `public string Title` – The title for the Hulu video.
- `public string EpisodeTitle` – The episode title for the Hulu video.
- `public int EpisodeNumber` – The episode number for the Hulu video.
- `public int SeasonNumber` – The season number for the Hulu video.

- `public string` `embedded` – The embedded string for the Hulu video.
- `public string` `Url` – The url to the Hulu video.

Constructors

- `public` `HuluParser(string input)` – Creates a `HuluParser` with the url that is to be parsed.

Methods

- `public bool` `parse()` – Runs the parser on the url that it was created with. Returns false if parse fails.

3.4.4: YouTubeParser

Parses the data from a YouTube page that has a video.

Properties

- `public string` `Title` – The title of the YouTube video.
- `public string` `Embedded` – The embedded string for the YouTube video.
- `public string` `Url` – The url for the YouTube video.

Constructors

- `public` `YouTubeParser(string input)` – Creates a YouTube parser using the url passed to it.

Methods

- `public bool` `parse()` – Runs the parser on the url that it was created with. Returns false if parse fails.

Appendix A: Coding Standards

Part 1: Syntax

For the most part AVM uses the same syntax that is normal by default in Visual Studio. This means 4 space indents instead of tabs when indenting. Also when a new block of code is created the { should be on its own line not following code. This rule has one exception that being that if you write a block of code that is one line long (in a property for example) you can leave the { on the same line as the rest of the code. Also each class is to have its own file, there should never be more than one class in a file. One should try and pass variables to and from a Class only via properties and there should be no public variables within a class, instead they should be private variables with _ in front of their name which are passed through a property. Parameters should be one to a line in order to make them easier to read. When calling classes within the AVM.Types or AVM.Parsers namespace make sure to keep the full path in front of the class name.

Part 2: Naming Scheme

The name style for AVM is very simple and as follows:

- Class names: Pascal Case (exp: AboutBox)
- Function Names: Camel Case (exp: refreshGroups)
- Property Names: Pascal Case
- Variable Names: Camel Case (if variable is also a property it should have _ in front of it example: _database)

Appendix B: File Listing

Folders

- src – contains all of the source files for AVM
 - Types – contains the classes that have been created for storage purposes
 - Parsers – contains the parsers used by AVM these include parsers for Hulu and YouTube.

Files

In src:

- AboutBox – The basic about box which is shown when you click on the About button.
- Database.cs – Has all of the functions that interact with the SQLite Database.
- DeleteConfirmation.cs – The form that makes sure you really want to delete a video.
- GroupEditor – The component that allows you to name and rename groups.
- Importer – The Importer dialog that is shown when you click the Import button. It allows you to restore backups as well as allows you to import whole folders of videos.
- mainForm – The form that is first shown when the program starts up. This is where most of the work is done and what the user will interact with the most.
- NodeEditor – The editor that pops up when a new Video is added and when the user edits a video already in the database.
- NodeViewer – The NodeViewer pops up when the user double-clicks on a node. This displays the information about the Video as well as giving the option to play the video and get more information.
- Program.cs – Starts mainForm when the program is first run.
- Settings – The Settings window that pops up when you click the Settings button. It allows the user to change overarching settings.
- WebPlayer – The player that is used to watch YouTube and Hulu videos without having to open the user's browser (uses the WebBrowser control for most of its work).

In Parsers:

- BackupParser.cs – The class that controls the backup to xml functions both reading and writing.
- FolderParser.cs – The class that parses both Folders and Files it allows files to be parsed by a predefined pattern.
- HuluParser.cs – Parses a Hulu url for all the important information that is needed for AVM.
- YouTubeParser – Parses a YouTube url for the all important information that is needed for AVM.

In Types:

- EpisodeInfo.cs – The file for the EpisodeInfo class which stores all the episode related information.
- FileData.cs – The file for the FileData class which stores all the file related information.
- Group.cs – The file for the Group class which stores all the information pertaining to a group.
- Node.cs – The file for the Node class which stores all the information about a node.