Telemetry and Communication IP Video Player

Zachary L. O'Farrell[1]
Colorado State University, Fort Collins, Colorado, 80523

Aegis Video Player is the name of the video over IP system for the Telemetry and Communications group of the Launch Services Program. Aegis' purpose is to display video streamed over a network connection to be viewed during launches. To accomplish this task, a VLC ActiveX plug-in was used in C# to provide the basic capabilities of video streaming. The program was then customized to be used during launches. The VLC plug-in can be configured programmatically to display a single stream, but for this project multiple streams needed to be accessed. To accomplish this, an easy to use, informative menu system was added to the program to enable users to quickly switch between videos. Other features were added to make the player more useful, such as watching multiple videos and watching a video in full screen.

Nomenclature

MRL = Media Resource Locator
UDP = User Datagram Protocol
LAN = Local Area Network
VLC = Video LAN Client
C# = C-sharp (Programming Language)
IP = Internet Protocol
SQL = Structured Query Language
XML = Extendable Markup Language
GUI = Graphical User Interface
LSP = Launch Services Program

I. Introduction

During my internship for the summer of 2011 I worked on a video over IP media player for LSP Telemetry and Communication called the Aegis Video Player. Video over IP is a way of transferring video streams over a network connection, and any computer connected to that network can access those streams. The Telemetry and Communications group of LSP handles all of the video feeds associated with unmanned expendable launch vehicles. The main purpose of this project is to have a way to monitor rockets during launches. To meet the purpose, the player has to be easy to use so video streams can be changed quickly to monitor certain situations during a launch.

[1] NASA USRP Intern, LSP Telemetry and Communications, Kennedy Space Center, Colorado State University

II. Related Software

The idea for the Aegis Video Player came from other software already available for use that performs a similar task. The two main software related technologies used in the Aegis player were the VLC ActiveX plug-in and Microsoft Visual Studios with the programming language C#.

A. VLC ActiveX plug-in

The main component in the Aegis Player is the VLC ActiveX plug-in. VLC stands for Video LAN Client. The VLC Media Player is capable of playing different types of media, including several streaming protocols, which are used in the Aegis Player. ActiveX is a way for program components to be reused in different programs. For the Aegis Player, the VLC ActiveX plug-in was a way to get the functionality of the VLC Media Player, while creating a new program with our own specifications.

B. Microsoft Visual Studios C#

The development environment I used was Microsoft Visual Studios. This program provides a visual way to create GUIs and can be used with C#. C# was picked for the programming language because it is meant for creating applications with user interfaces and supports the VLC ActiveX plug-in.

III. Development

The Aegis Video Player's goal was to Receive and display IP Video, but it also had several other requirements to make the player usable during launches. These included an easy way to change the video streams, choose between watching one or two videos, quickly switching from one or two videos to one video in full screen, creating easy to read menus, and the ability to send commands to the player from another program. Not only is the Aegis Player to be used in launches, it is also meant for day-to-day use, so a requirement was added to add some advanced features. These features gave the user more control in using the Aegis Player but make the player more unstable and more likely to crash.

A. Menu GUI

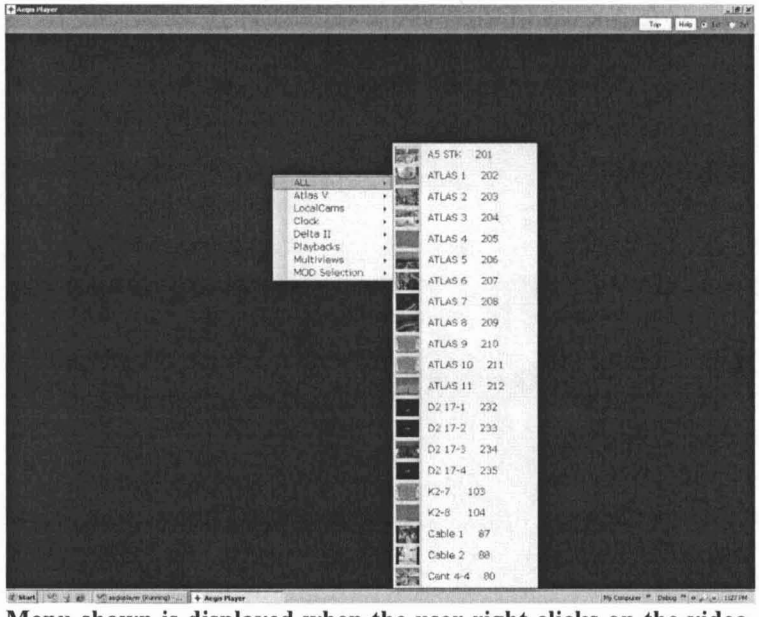
My first big requirement was to create a way to change videos that was fast and easy to use. My two main options were a pull down menu and adding a right click pop-up menu that displayed a list of videos. I tried two different ways to make pull down menus. Both worked well but required an extra tool on the form and were slower than the right clicking. The right click option was much more difficult to do, but proved to be the best way because of how easy it is for the user to use.

My first drop down attempt was a drop down menu at the bottom of the screen below the video. This was the easiest option to implement because Visual Studios comes with a tool for drop down menus. This worked well and provided a way to change the video, but the menu was always visible, even when the user wasn't trying to change the video.

My second drop down menu was designed to make the drop down menu invisible while the user wasn't trying to change the video. This was accomplished by only showing a small bar on the left side of the screen. When the user hovered over the bar, the menu popped out from the side bar allowing the user to select a video. I decided not to use this because it was too slow. The User had to hover over a small bar and wait for the menu to pop up. This method was better than the drop down menu, but it was not ideal. The main programming complication with this design was making the menu display at the correct time. I had to create events for hover, mouse over and mouse leave. When the mouse hovered over the side bar,

the menu needed to appear and it needed to stay until the user was done selecting a video. To make sure the menus were shown while the user was selecting a video, I had to use an on mouse enter event. This event worked by showing the menus when the mouse entered the menus. After a video was selected the menus went away but it the user did not click the menus still stayed visible on the screen. To overcome this I did a mouse leave event on the menu, so when the mouse left the menu, the menus were hidden.

After trying both of these designs I realized that the number of menus had to be equal to the number of videos. The requirement of this player was to have one or two videos playing which meant having one or two dropdown menus displayed at one time. This was reasonable because of the small number, but in the future, as computers get faster and are able to support more videos, the number of videos will continue to grow and the number of menus with them. I felt that even doing four separate menus would make the screen too cluttered with four videos and a menu for each one. The right click method allowed just one menu for in infinite number of videos and only one instance of the menu could be brought up at any one time. Despite being more difficult, the right click was the best solution because it satisfied all the criteria for providing a way to change the video source. Using right click allowed the user to click anywhere on a video, whereas drop down menus required the mouse to be at a certain position. Most tools in C# provide an event handler that deals with right click, however the VLC ActiveX plug-in does not support mouse events. When the plug-in is playing a video, small windows, that are invisible to the user, pop-up and take focus away from the player, because the pop-up windows have focus, all mouse events are sent to them instead of the plug-in. Currently, the VLC plug-in does not provide a way to capture the clicks on the pop-up windows, nor is there a way to take the messages sent to the pop-up windows and forward them to the plug-in.



Menu shown is displayed when the user right clicks on the video. Accomplished by overriding the "wndProc" method and intercepting the message sent by Windows for right click. The same menu appears for each video, but the video changes depending on the mouse location.

I tried several different workarounds to overcome the lack of the ability to right click on the video. My first attempt was to send the mouse events to the main form. The main form is the base window where all of the tools, such as buttons, labels, and the VLC ActiveX plug-in, are located. The problem with this method is that when the mouse enters another tool, such as the VLC plug-in, the focus switches from the main form to the specific tool. After implementing this idea, the Player accepted right clicks on the background of the forms, but not on the video.

My second attempt was to add a tool on top of the plug-in, make it transparent, and send mouse events to it. For this I used a panel tool. A panel is a tool that is used to group multiple tools together; it allows both mouse events and has a transparent setting that makes it invisible to the user. This did not work because tools that are transparent send all events to their parent forms. So when no video was playing, the

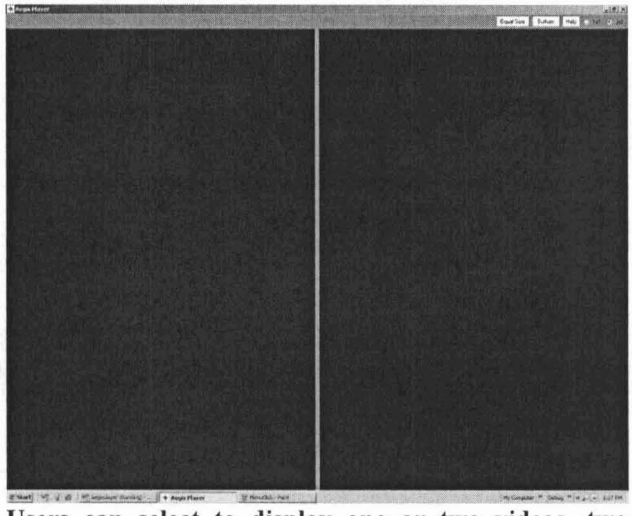
panel sent mouse clicks to the main form, but if there was a video playing the mouse event was sent to the plug-in and then lost to the pop-up windows.

My third attempt was to intercept the mouse click messages on the whole form before anything processed them. I was able to do this by overriding the method "wndProc". "WndProc" stands for Windows Procedures, which handles everything from appearance, to moving the window, to mouse events. Overriding this method allows all messages sent through this method to be intercepted and redirected. I was able to use this by checking all of the messages that were sent through and comparing them to the codes for a right mouse click. Messages were only received by the program when it had focus, so no other windows features where affected by overriding the Windows Procedure method. After I had overridden the method, I was able to detect right clicks and then display a pop-up list of videos to play where the mouse was. A problem with this was that the right click worked anywhere on the screen when I only wanted it to work when the mouse was over a video. To solve this I put in a mouse position test after receiving a message, and if the mouse wasn't within the borders of a plug-in the right click event did not trigger.

After several attempts, the right click proved to be the best solution to changing the video because it does not require anything to be put on the screen as other methods do, and it is easy to use. The right click fulfilled the requirement of a fast, easy way to switch between videos.

B. Number of Videos

A goal of the Aegis player is to be able to switch between one and two videos. More videos would be preferable but computer resources limit the number of videos that can be played at one time. Playing more than two videos at one time uses over half of the CPU; this is unacceptable because during launches, there are other programs running on each computer that need CPU as well. While the player is running, the user can switch between one and two videos by selecting a radio button.



Users can select to display one or two videos, two video option shown. One can be played to the left of the blue line and one to the right. The middle bar can be resized to make one video bigger and the other smaller while keeping the window the same size. Double clicking one of the videos stops the other one and makes the selected one go full screen.

Only allowing two videos on the player limits the CPU usage, but if enough windows are opened the player can crash the computer. To solve this I added a "mutex". "Mutex" stands for mutual exclusion. A mutual exclusion is a way to keep track of computer resources and ensure that something is not being used twice. The Aegis player uses this to look at the processes running on the computer and determine if the Aegis Player is running or not. If there is already an instance of the player running, the newly opened one is closed and the old one is brought to the front of the screen, otherwise, a new Aegis Player window opens.

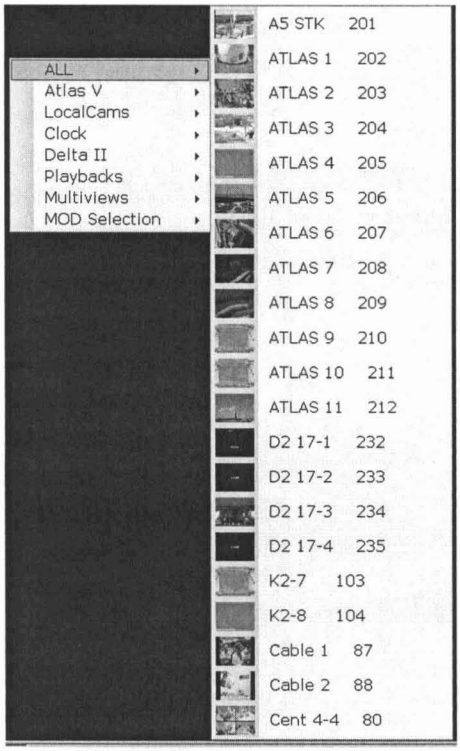
The Aegis player allows multiple videos to be viewed while still limiting the resources. Both of these are important because it allows users to monitor multiple videos and still run process in the background.

As the computers used become faster and better at handling video streaming, more videos will be added to the player.

C. Double Click

Another requirement of the player was inspired by the VLC Media Player, double clicking a video to go full screen. The idea is that if someone is watching one or more videos and they want to make just one go full screen, they can double click on it and that video will go full screen and when they double click again the video returns to its previous size and continues playing the videos it was playing. When I tried implementing this I had already enabled the right click feature, so I used the "wndProc" method that was already implemented. After looking at the messages sent by through the windows procedure method, I realized that the system did not send a double click message.

To overcome not having a double click message, I used the left click message and a timer. When the left mouse button was clicked, I started a timer. The next time the left mouse button was clicked I stopped the timer and checked the time. If the time was less than a specified time, the mouse position was checked and the video that the mouse was over went full screen. If it was greater than the specified time, a normal left click event was fired and the timer restarted.



Menu shown uses categories on the first popup window. Moving the mouse over any category opens a menu that contains all the videos in that category. Menu shown has the "All" category open, and all of the video sources are shown to the right of the category menu.

D. Usable Menus

After creating a way to easily access a menu of videos, I had to make the menus usable. The first menus I used just showed the MRL. MRL is the address of the media to be played; this is similar to a URL but gets Media instead of generic information. This is useless for most people because it doesn't show what video is actually playing on that source. Hard coding names into the menus were not possible because they change so much. To get a useable name for each video I had to make a call to a database using SQL. In the call I got the name of the video, the number of the video, the MRL, a thumbnail, and an array of categories. I put all of this information into a class that the menus could access to display the correct information. After creating the menus, each source displayed a thumbnail, the name of the video, and the number of the video. To further simplify the menu, each source was put into one or more categories, such as "All" that contained all of the videos and "Weather" that had videos related to weather. Categories could be added to or removed from each video through the database. The categories were set as the first layer of the menus. On right clicking, a menu popped up that listed all of the categories, by moving the mouse over a category another menu popped up that displayed all of the videos in that category.

The limitation of using a database is that if the database is not working the player becomes unusable. To overcome this I used a XML configuration file. The configuration file contains the call to the database, as well as the MRL for each source and a name that can be entered. By doing this, if the database is down, or if there is no database, someone can go into the XML file and change the MRL and

name of each source. If the database is not found, the program uses the XML file to create the menus by default, these menus are less useful than the ones created by the database, but they allow the player to be used if there is a problem with the database. The advantage of using an XML file is that is more readable than the Aegis Video Player code. The XML file contains less than fifty lines of code whereas the Aegis Player has more than a thousand lines. So it is much easier to change the XML file than it is to change the code in the program.

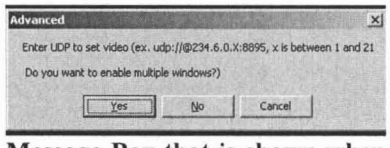
E. Mail Slots

Iris is a program written and used by LSP that can display information about launch vehicles. One feature of Iris is being able to send commands to different programs. The Aegis player was added to the list of programs that Iris can send commands to. Iris can send messages to the Aegis player that can change the video currently playing, and start the program remotely.

The sending and receiving of messages is done through mail slots. A mail slot is a file created by one program that can be written to by another. The program that creates the file can only read the mail slot, and other programs can only write to it. The mail slot on the server side (the side that can read) is put on a separate thread from the rest of the program and periodically checks the mail slot for any messages. When a message is received it is in the form of an array of bytes, the program then decodes it to a string and processes it. When it is determined what the command is, the player executes the command. Commands can be anything, but for the Aegis player they are limited to what video is playing and starting the player.

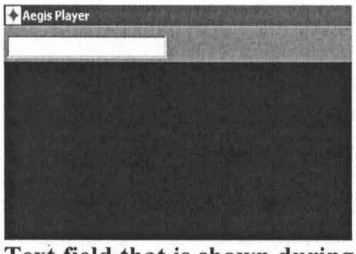
F. Advanced Features

Although the main goal of the Aegis Video Player is an IP Video player that can be used during launches, there is also a need to use it in day-to-day activities. These include testing video equipment and continuous monitoring of videos. To accomplish this, some advanced features were added to the player. Most of the advanced controls remove features that make the player more stable. These are only intended for people who know the limitations of their computer and understand that the player can crash their computer if they overuse it. I used two separate methods for using advanced controls, a series of command line arguments and a keyboard shortcut that could be used while the program is running.



Message Box that is shown when the keyboard shortcut is used on the player. Users have the option of turning off single instance operation or not, and canceling the operation.

Command line arguments are used when running the program from a command prompt and are input as strings from the user. The strings are then sent to the program to be sorted through. Each string is tested to contain certain words, phrases, or characters and then a command is executed based on the string. For example, if a string is input through the command line the program looks for the substring "full", if it is found the player opens in full screen mode, if it is not found other tests are run, if no matches are found the string is ignored. The main command is the ability to disable only opening one instance of the program. This is accomplished by checking for the argument "disable", if it is found the player runs whether or not another instance of the program was opened. Other features include specifying how many videos to play, the MRL of each, the size of each video if there are two videos, and making the window stay on top of other windows, even when loosing focus.



Text field that is shown during advanced mode operation. Allows user to select a stream to watch by entering the MRL.

A keyboard shortcut to use some of the advanced features was added because command line arguments are only used when the program starts and from the command prompt. The advanced features are used by pressing the control key and the 'D' key. When the advanced features are enabled this way, a pop-up box appears and asks the user if they want to disable single instance operation of the application. If they choose yes, another instance can be opened, if no then another instance cannot be opened. Either way, a text field appears that allows user input. The text field allows the user to type in a MRL for each video playing to change the video playing.

IV.Conclusion

At the end of my ten week internship I successfully created an IP video player that would be used during LSP launches. This project had several requirements that I had to meet to accomplish the goal of the player, which was to make a fast, easy to use video over IP player. Aegis has yet to be used on a launch yet, but it is ready to be used pending the installation of the software on the launch consoles. This project allowed me to learn several new skills. I learned C#, which I had never programmed in before, how to use Microsoft Visual Studios, how to make SQL database calls, and how to create and use XML configuration files. Although I did not need the information to do my project, I was able to learn about some of the hardware involved in transporting video over an IP network. Overall, my project was a success because I was able to accomplish the goals of the Aegis Player and learn about the hardware and software related with IP video.