Channels
Quick Easy Video Content Consumption

Alexander Ledwith
California Polytechnic State University, San Luis Obispo
June 2013

©Alexander Ledwith
Table of Contents

Introduction .......................................................................................................................... 4
Project Overview .................................................................................................................. 4
Need Statement .................................................................................................................... 4
Project Goals and Objectives ............................................................................................ 4
Project Outcomes and Deliverables .................................................................................. 5

Formal Project Definition ................................................................................................. 6
Marketing Requirements ................................................................................................. 6
Engineering Requirements .............................................................................................. 6
Architecture Overview ....................................................................................................... 8

Playing Content ................................................................................................................ 9
  Architecture ........................................................................................................................ 9
  Loading Content .............................................................................................................. 10
  User Interface .................................................................................................................. 10

Acquire Content ............................................................................................................... 13
  Architecture .................................................................................................................... 13
  Metadata Manager .......................................................................................................... 14
  User Interface .................................................................................................................. 15

Implementation ................................................................................................................. 15
System Requirements ....................................................................................................... 15
Backend .............................................................................................................................. 16
  Database .......................................................................................................................... 16
  Video Item Model ........................................................................................................... 16
  Schedule Item Model ....................................................................................................... 16
  Channel Model ................................................................................................................ 17

Playing Content ................................................................................................................. 17
  Channel Switching View ............................................................................................... 17
  Video Playing View ........................................................................................................ 21
  Channel Switching View Table ...................................................................................... 22

Acquiring Content ............................................................................................................. 22
  Display Content Window ............................................................................................... 22

Future Implementation ....................................................................................................... 23

Testing and Validation ...................................................................................................... 23

Conclusion .......................................................................................................................... 25
Table of Figures

Figure 1: Architecture Overview Diagram .......................................................... 8
Figure 2: Playing Content Architecture ............................................................. 9
Figure 3: Video Playing Interface Mockup ......................................................... 10
Figure 4: Channel Switching View Table Mockup ............................................. 11
Figure 5: Channel Switching View Library Mockup ......................................... 12
Figure 6: Channel Switching View Settings Mockup ....................................... 12
Figure 7: Acquire Content Architecture ......................................................... 13
Figure 8: Acquire Content User Interface Mockup .......................................... 15
Figure 9: Backend Relational Database ......................................................... 16
Figure 10: Channel Switching View User Interface ......................................... 18
Figure 11: Video Playing View User Interface .................................................. 21

Table of Tables

Table 1: Engineering Requirements ................................................................. 7
Table 2: Small Viewing Mode Commands ....................................................... 20
Table 3: Full Viewing Mode Commands .......................................................... 20
Table 4: Supported Video File Types ............................................................... 23
Table 5: Testing Results ................................................................................. 24
Introduction

Project Overview

The idea for this project is to take a user’s digital movie and television show library, and organize the individual videos into lists that will continuously and concurrently play like cable TV channels. This means that when a list of channels is switched to, video content will automatically start playing, possibly in the middle, based on a schedule. It should serve the needs of the user by allowing the user to quickly watch any of their existing channels, easily add a new channel based on chosen criteria or manual selection, and easily add new video content. In addition, the product should automatically fill in metadata for all video content. Design decisions must be made in the following categories: user interface design, memory usage of currently playing videos, metadata fetching efficiency and accuracy, and automatic channel creation algorithms.

Need Statement

There is currently two main ways users consume professionally made video content: cable television, and online distribution. Both methods have their pros and cons. Cable television allows users to very quickly begin watching video content. It does not however, guarantee content quality based on user preference. Online distributed video allows users to choose content, guaranteeing the content fits the preference of the user. Unfortunately, to allow the user to choose specific content requires more overhead. The goal of this project is to provide users with the low overhead cable television offers, while increasing the quality of the content.

Project Goals and Objectives

Goals:

- Create an easy-to-use, intuitive interface that allows user to quickly view their video content.
- Create an easy way to switch between channels.
- Allow the user to easily add content to the application.
- Automatically fill the metadata of all the content.
- Allow the user to create channels based on given criteria from the metadata.
- Allow users to manually create their own channels.
- Allow users to view and edit their current video library.

Objectives

- Begin implementation in the beginning of April 2013.
- Create usability tests from the beginning to test the product.
Project Outcomes and Deliverables

The outcome of the project will be a fully functional Mac desktop application. This application will come in a single package that is easily distributed. In addition, the source code for the product will be included in an accessible repository.

Project Duration:

Research - 1/7/2013 - 3/29/2013
- Prior Projects: Find similar video playing projects to see how the provide the stored video content to the user.
- User Interface: Look into applications that display video and compare the user interfaces.
- Metadata Fetching: Find the services that provide metadata for professionally made video content. Find the tools to fetch the metadata for this application.
- Automatically Organizing Data: Look into algorithms to automatically organize data based on similarities.

Design - 2/25/2013 - 4/1/2013
- System Architecture: Design how all the pieces of the project fit together. The architecture should be in accordance with the specified goals of the project.
- Database Design: All channel information and metadata is going to need to be stored in a database.
- Channel Switching/Efficiency: Multiple channels are going to need to be loaded at the same time so it is important to load those channels in an efficient way.
- Adding/Removing/Organizing Video in Video Library: The user will need a way to organize content in the library.
- User Interface: Design the look and feel of the product with usability in mind. Create specification on how the user interface should perform in given situations.
- Metadata Fetching: The modules used to fetch the metadata need to work in the background efficiently.
- Automatic Channel Creation: The ways in which channels can be automatically created needs to be considered.

Implementation - 4/1/2013 - 5/24/2013
- Each piece in the project will be implemented in the current order listed in the design section: system architecture, database models, channel switching/efficiency, user interface, metadata fetching, and automatic channel creation.

- Functional Testing: the first step in testing is to make sure that each piece works in the common, designed for scenario.
- Edge Case Testing: next all edge cases need to be tested.
- Usability Testing: Once the product is believed to work as described, then it needs to be tested for ease of use, and intuitiveness.

Revision 6/3/2013 - 6/7/2013
- Once all pieces have been tested the product should be revised with the needed improvements that were discovered during the testing phase.
Formal Project Definition

Marketing Requirements
1. Simple/Intuitive User Interface
2. Easy Content Addition
3. Metadata Can Be Filled
4. Quick Channel Switching
5. Schedules Can Be Created
6. Channels Can Be Created
7. Does Not Use a Huge Amount of Memory
8. Does Not Use Excess Processing Power
9. Visually Appealing
10. Easy to Use
11. Interface Responds Quickly

Engineering Requirements

<table>
<thead>
<tr>
<th>Marketing Requirements</th>
<th>Engineering Requirements</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 4, 11</td>
<td>Channel switching takes less than 2 second.</td>
<td>Users will not want to wait while flipping through channels.</td>
</tr>
<tr>
<td>1, 9</td>
<td>Video controls disappear after 3 seconds of inactivity.</td>
<td>This will allow users to enjoy their video content without distraction.</td>
</tr>
<tr>
<td>1, 2, 10</td>
<td>Users should be able to add content in less than 1 minute.</td>
<td>Content addition is integral to having content to display.</td>
</tr>
<tr>
<td>3</td>
<td>Metadata will be fetched when a piece of content is added.</td>
<td>This way all content will have filled in metadata.</td>
</tr>
<tr>
<td>5</td>
<td>Schedules will be asynchronously created once timing runs out.</td>
<td>Schedule should always be up to date and its creations should not slow down the main thread</td>
</tr>
<tr>
<td>6, 10</td>
<td>Channels can be created manually</td>
<td>Users should be able to design channels their way.</td>
</tr>
<tr>
<td>6, 10</td>
<td>Manually created channels can be created within less than 1 minute</td>
<td>Should not be too much overhead to channel creation</td>
</tr>
<tr>
<td>6, 10</td>
<td>Channels can be created</td>
<td>Users should not have to</td>
</tr>
<tr>
<td></td>
<td>automatically based on metadata fields</td>
<td>create channels manually if they do not want to.</td>
</tr>
<tr>
<td>----</td>
<td>--------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>The system will use no more than 1GB of memory</td>
<td>The system should not negatively affect other processes running on the computer.</td>
</tr>
<tr>
<td>7</td>
<td>Each video file will be able to hold buffered video in a container with max length of 100MB</td>
<td>No video file should take up too much space on the hard drive</td>
</tr>
<tr>
<td>8, 11</td>
<td>No more than four videos should be buffering at the same time.</td>
<td>The system should not use a huge amount of processing power.</td>
</tr>
<tr>
<td>9, 10</td>
<td>The User Interface will incorporate a dark feature design with no more than 3 main color types.</td>
<td>A consistent user interface will allow for a more pleasing experience for the user.</td>
</tr>
</tbody>
</table>

Table 1: Engineering Requirements
The application will consist of two main sections: the playing content section and the acquiring content section. The playing content section will handle all channel scheduling, channel loading/unloading, rendering the video that is to be played, and handling user events that pertain to changing channels. In addition, the playing content section will also allow for quick creation and modification of channels. Finally the playing content section will allow the user to adjust settings that pertain to that section. The acquiring content section will handle acquiring all new content for the application, organizing the current content, and fetching metadata for both new and existing content. In addition the acquiring content section will also have the ability to manage channel creation.

Both sections will interact with the central database through the channel and video item models. Each section will have an active role in both reading and writing to the database. The acquiring content section will display current content by reading the existing content from the database. It will also write to the database when new content has been added, or metadata information has been updated. The playing content section will read from the database to acquire the content to play in the form of individual content, and the organization of the content based on channels. It will write to the database when it creates a new schedule for a given channel or when the user updates a channel.
The Main Application Controller will handle all global application needs such as: starting and stopping the application, switch between two sections, and handling global user preferences.

## Playing Content

### Architecture

The overall architecture for the playing content section consists of one main Channels Controller, and two main views: video playing view and the channel switching view.

The main Channels Controller handles organizing all the channels that are available. This organization includes keeping track of which channel is the current channel, interpreting schedule information to let the Video Playing Controller what to play next, and the switching between views. To obtain the available channels the Channels Controller communicates with channels models to fetch its array. In addition to communicating with the database to get the list of channels, the Channels Controller must also let the channels models know when to start loading or stop loading their video data.

The Video Playing Controller handles the view when the only thing displayed is the video. It handles rendering the video, and user input to play, pause, fast forward, rewind, and skip to the next video in the channel. The Video Playing Controller also allows the user to switch to adjacent channels in the current ordering of channels.

The Channel Switching Controller handles when the user wants to view all the channels that are available, or switch to one. This controller also handles displaying the current video library, and updating existing channels.
**Loading Content**

The Channels Controller holds the control over when data is loaded. The Channels Controller has an array of all the available channels. It also knows which channel is currently playing, and what channels are adjacent to that channel. Adjacent channels are the ones that are most likely to be switched to because the Video Playing Controller can switch to these with a single user input command. Knowing this the Channels Controller can decided which channels to be actively loading at any given time. The actual loading of the channels relies on both the Channels Model and the Video Item Model. The Channels Model has an array of Video Items, and it knows which Video Item is the current one. Using this information the channel model can tell the current video item to begin loading. The actual loading of the video file happens within the Video Item model itself. The initial metric is to have no more than four video files being loaded at the same time. This value is subject to change based on performance tests.

**User Interface**

*Video Playing View*

![Video Playing Interface Mockup](image)

The video playing view consists of two aspects: the actual content that is playing, and the controls that can modify that content. The explanation of each control is as follows:

- **Menu** - transitions the application to the channel switching state.
- **Rewind** - sets the play rate of the content the content to -2.0 and updates the underlying schedule accordingly.
- **Play/Pause** - sets the play rate of the content to 0 and updates the underlying schedule accordingly.
- Fast Forward - sets the play rate to 5.0 and updates the underlying schedule accordingly.
- Skip to Next Video - loads the next scheduled video content and updates the underlying schedule accordingly.
- Change Channel - changes the channel to the next/previous channel in the channel list.

**Channel Switching View**

<table>
<thead>
<tr>
<th>Channel</th>
<th>Currently Playing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comedy</td>
<td>Funny Movie</td>
</tr>
<tr>
<td>Action</td>
<td>Action Movie</td>
</tr>
<tr>
<td>Drama</td>
<td>TNT Knows Drama Show</td>
</tr>
<tr>
<td>Thriller</td>
<td>Michael Jackson's Thriller</td>
</tr>
<tr>
<td>Favorite</td>
<td>I Love This Movie Movie</td>
</tr>
<tr>
<td>Community Choice</td>
<td>Everyone Loves This Movie Movie</td>
</tr>
<tr>
<td>All Videos</td>
<td>Everything I Own Movie</td>
</tr>
</tbody>
</table>

*Figure 4: Channel Switching View Table Mockup*
Figure 5: Channel Switching View Library Mockup

Super Cool Movie
This is the description of the Super Cool Movie, could include summary, actors, directors, rating, etc.

Figure 6: Channel Switching View Settings Mockup

Volume

30%

Resolution

1080p

Brightness

75%
The channel switching view is split into three tabs: channels, library, and setting. The channels tab shows the list of all the user's channels so they can choose which one they want to switch to. By holding down any given channel it will expand to show the upcoming videos for that channel. The library view shows all video content that is in the user's library. By selecting a video from the library view, that video will expand to show more information about the video, allow the user to simply play the video, and allow the user to add that video to an existing channel. The settings view displays generic settings pertaining to video watching.

**Acquire Content**

**Architecture**

The acquiring content section is centered on the Display Content Controller. The Display Content Controller shows the whole window for this section. It allows the user to choose what type of management task they want to perform. The Display Content Controller contains an array of all the video items that have been added to the application, and it contains an array of all the channels that have been created. It can load up two different views: the Display All Video Items view, and the Display Channel Content view. The Display All Video Items view shows every Video Item that has been added to the application, everything that is contained in the corresponding array. The Display Channel Content view will only show the Video Items that are contained within the chosen channel.
In addition to displaying content, the Display Content Controller allows the user to access the Add Video Content Controller. The Add Video Content Controller allows users to add new video content to their library. This is done by allowing users to search their local machine for files they would like to add to their video library. Once the file is chosen a new Video Item is created.

When a file is added to the library, the Metadata Manager processes it. The Metadata Manager determines if the video is professionally made and has publicly available metadata information. If the video does have publicly available metadata information, the Metadata Manager fetches that information. Once that information is fetched the manager stores the information into the database and associates it with the newly added Video Item that was created when the file was added.

**Metadata Manager**

There are currently many different services that store metadata information about both movies and television shows. Along with the different services come different APIs to access the stored information. As part of the implementation of the Metadata Manager, the services and API will need to be compared to see which ones fit well with this product.

Before calling the chosen fetching APIs the name of the content is going to need to be parsed from the file name. Many services are somewhat picky about how the content is searched for. To pull out the correct name a parsing algorithm will be implemented. This parsing algorithm will use regexes to pull out possible names from the given file name, and send those possible name over to the metadata service. It will then make the decision on which result is the best.

In addition to the parsing algorithm the user will have the option of manually entering in the name of the content and other identifying information. This will allow the user to have more control over the metadata that is fetched.

Once the metadata is fetched it will be stored in the database and associated with a Video Item.
The display content user interface is split into four parts, the header, the footer, the sidebar, and the main content display. The purpose of the header is to display options for how the main content is displayed. The options are, picture only, picture and description, or straight list. The footer allows the user to edit content in the main content section by adding, deleting or editing content. The sidebar allows the user to switch between simply viewing content in their library to viewing their current channels. The main content section shows the Video Items for the selection of the sidebar.

### Implementation

This project was implemented as a Mac OS X application using the Mac development platform Cocoa. Cocoa is a set of APIs on top of the Objective-C language. It provides the access to many of the underlying Mac services.

### System Requirements

- Intel based Mac Computer
- OS X 10.8 Mountain Lion
- 2GB of RAM
- 10 MB of Hard Drive Space
Backend

The backend for the project has the important task of storing all the persistent data of the application. It contains four main parts: the database and database interaction system, the Video Item model, the Channel model, and the Schedule Item model.

Database

The decision to use a database system is based on the ability of databases to store large amounts of data efficiently. An SQLite database is used in conjunction with the Cocoa technology Core Data. Core Data is an object graph management system that allows the developer to easily create complex object relations. It can then use the designed object relations to create a relational database in a chosen format. Because Core Data handles the object graph and the database, it provides a layer between the developer and the database. This layer makes updating and saving information to the database from the current object graph easier.

Video Item Model

The Video Item model is mainly a container model to hold all the information about a given video in the user library. It contains the video files location, and the following pieces of metadata: video duration, name, image, and description. It also contains the relationships to the Channels that include it and the Schedule Items that use it.

Schedule Item Model

The Schedule Item model is used to contain information about each an item in a given Channel’s schedule. It contains its play position in time with a start and stop date, a reference to the Channel that owns it, and a reference to the Video Item it contains. It also contains a specialized method that determines whether the current time is contained in its time range.
Each time a new Video Item is added to a Channel a new distinct Schedule Item is created. That newly created Schedule Item is then added to the schedule for the Channel.

Channel Model

The Channel model contains all the channel information and specialized methods for interacting with that information. The key components the Channel model contains are as follows: the list of all its Video Items, an ordered list of Schedule Items that constitutes the current schedule, its name, its number in the list of channels, and its current Schedule Item. All of these components are statically stored except for the current Schedule Item. The current Schedule Item is dynamically calculated. It depends on the current time of day, and what range out of all the time ranges in the schedule that the current time of day fits into. The Channel model calculates this using the following steps:

1. If the current time falls in the range of the previously used Schedule Item return that item.
2. If the current time is past the range of the whole schedule, shuffle the schedule and return the first Schedule Item.
3. If the current time is in the whole range of the schedule, binary search to find the correct Schedule Item.

Playing Content

The Playing Content section of the application is the main section the user will interact with. It handles management of video playing, channel switching, view switching, and user input through the Channel Switching View. It handles video playback controls, and viewing experience through the Video Playing View. Finally it handles displaying channel schedules with the Channel Switching View Table. Putting all these pieces together allows the user perform the two main functions of the application: watching video and switching channels.

Channel Switching View

The main function of the Channel Switching View is to handle requested updates from other parts of the system including its sub views and the input devices. The two sub views it responds to are, the Video Playing View and the Channel Switching View Table. It also responds to keyboard, mouse and Apple Remote ® events. Finally it responds to updates from the Display Content Window. By being a top-level object that connects to multiple parts of the application, the other views can direct their requests to the Channel Switching View and it will handle that request appropriately.
The interface shown in Figure 10 is the interface when the Video Playing View is in small viewing mode; the video playing view is located in the upper right corner of the frame. The Video Playing View can also be in full viewing mode, where it covers the complete frame of the Channel Switching View. When in small viewing mode, the Channel Switching View Table and the currently playing video information is visible.

All the pieces in the user interface are laid out using the cocoa interface layout system Auto Layout. Auto Layout is based on setting up constraints between views. In most cases it is used to statically constrain views to specific regions in a parent view, but it can also be used to create animations of a view. This type of Auto Layout based animation is used to animate the switching between the small viewing mode and the full viewing mode. The constraints that constrict the Video Playing View to the upper right corner can be changed and animated to constrict the Video Playing View to be the same size as the Channel Switching View.

Channel Switching View Table Interaction

The Channel Switching View handles updates from and sends updates to its table. The main update that the table sends to the Channel Switching View is when a table cell has been selected. A selected table cell indicates that the user want to switch to the selected video from the channel. Once the Channel Switching View receives this update it determines which channel and video the table cell represents and then it switches the currently playing video in the Video Playing View. The Channel Switching View also lets its table know when it needs to update. The times when the table will need to update include: when the Video Playing View requests either a current channel change, a change in the channel time ranges, or a switch to the next video in a channel. Updating is also needed when the Display Content Window changes or adds a channel.
**Video Playing View Interaction**

The Channel Switching View handles the current channel and the currently playing video in that channel. This handling means that whenever the Video Playing View receives user input to modify the currently playing content, it must notify the Channel Switching View. These modifications can include: any kind of playback change (e.g. pause, fast forward, rewind, etc.) that will cause the Schedule Item’s time range to change, channel switching up or down, switching to the next video in the channel, switching to small viewing mode was requested, or the currently playing video finished playing. If the request received by the Channels Switching View requires changing the currently playing video, the Channel Switching View chooses the next video and then sets that video in the Video Playing View.

**Keyboard and Apple Remote ® Interaction**

The application makes extensive use of the keyboard commands to handle user input. Each view handles its own set of keyboard command, but many times the command is sent up to the Channel Switching View to handle. For example when the return key is press on in the Channel Switching View Table it forwards that message on to the Channel Switching View. Tables 2 and 3 displays a full list of keyboard and remote input commands.

The application uses an open source framework call HIDRemote to handle the Apple Remote ® input. This framework handles setting up the remote interaction, preventing other applications from intercepting remote signals, and detecting which buttons have been pressed. It can also detect what type of Apple Remote ® is being used and whether or not the button presses were clicks or holds. Once the application detects a remote button press, it determines what action from the already set up keyboard responses to take.

**Small Viewing Mode Commands**

<table>
<thead>
<tr>
<th>Keyboard Command</th>
<th>Remote Command</th>
<th>Action</th>
<th>Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC</td>
<td>Menu</td>
<td>Toggle to full viewing mode.</td>
<td>Channel Switching View</td>
</tr>
<tr>
<td>ENTER</td>
<td>Play/Pause</td>
<td>Switch to currently selected video item in table.</td>
<td>Channel Switching View</td>
</tr>
<tr>
<td>Arrow Key Up</td>
<td>Up Button</td>
<td>Select table cell above current cell. Roll over at top.</td>
<td>Channel Switching View Table</td>
</tr>
<tr>
<td>Arrow Key Down</td>
<td>Down Button</td>
<td>Select table cell below current cell. Roll over at bottom.</td>
<td>Channel Switching View Table</td>
</tr>
<tr>
<td>Arrow Key Right</td>
<td>Right Button</td>
<td>Select table cell above after cell. Roll</td>
<td>Channel Switching View Table</td>
</tr>
<tr>
<td>Keyboard Command</td>
<td>Remote Command</td>
<td>Action</td>
<td>Handler</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>ESC</td>
<td>Menu</td>
<td>Toggle to small viewing mode.</td>
<td>Channel Switching View</td>
</tr>
<tr>
<td>ENTER</td>
<td>Play/Pause</td>
<td>Play/Pause currently playing video</td>
<td>Video Playing View</td>
</tr>
<tr>
<td>Arrow Key Up</td>
<td>Up Button</td>
<td>Change channel up.</td>
<td>Video Playing View</td>
</tr>
<tr>
<td>Arrow Key Down</td>
<td>Down Button</td>
<td>Change channel down.</td>
<td>Video Playing View</td>
</tr>
</tbody>
</table>

**Table 3: Full Viewing Mode Commands**

*Display Content Window Interaction*

The Display Content Window handles all the organizing of the Video Items and the creation and deletion of the channels. Once a channel has been changed in the Display Content Window the Channel Switching View will need to be updated to reflect the changes. The Channel Switching View handles two different types of updates from the Display Content Window. The first is a direct notification when a channel is going to be deleted incase the channel is being used. The second is an indirect notification when a channel is updated. The indirect notification is used to prevent the Display Content Window from having to wait while the Channel Switching View updates.
**Video Playing View**

The main goals of the Video Playing View are handling playing the video content, and handling user inputted playback commands.

**User Interface**

![Figure 11: Video Playing View User Interface](image)

The user interface of the Video Playing View looks to maximize the viewing experience. Most of the time only the video content is displayed on the screen. If the user does want to adjust aspects of the playback they can move the mouse to make the controls appear. The controls include the following adjustments: play/pause, fast forward, rewind, switch to the next scheduled video within the channel, switch to the previous channel, switch to the next channel, and change to small viewing mode. If there is no user activity for three seconds after the controls appear the controls fade away.

**Playing Content**

To play content the application uses an open source project called JKSMoviePlayer that sets up standard video playback and controls. JKSMoviePlayer uses the audio and video Cocoa Framework AVFoundation to load, render, and play content. AVFoundation provides a multi-part system that allows for flexibility when playing content. The main AVFoundation components used in this application include: the asset item which loads the file, the player item which provides content to the player, the player layer which handles displaying content, and the player which handles playback. The player layer is the most expensive component to create, so a player layer for a given video item is only created once and then stored. A dictionary is used to store a mapping between the video item and the stored player layer.
Channel Switching View Table

The Channel Switching View table handles displaying all the user’s channels in a tabular format. It displays the channel name and number, and three Schedule Items from the channel. The three Schedule Items that are shown are as follows: the item whose time range matches the current time, the item right after the current item, and then the second item past the current item. This setup gives the user a quick visual representation of what videos they can look forward to, and what channels they can switch to.

User Interface

The user interface uses a customized NSTableView. NSTableViews are the standard Cocoa API for creating tables. This application makes a number of adjustments to the table view to customize the look and feel of the table. First it creates two different types of cells for the table view. The first cell is the channel name cell. The channel name cell contains a specially formatted image with the channel number, and a label to display the channel name. The second cell is the schedule item cell. The schedule item cell contains three items: the Video Item’s image, the Video Item’s name, and the Schedule Item’s time range. The next customization is the black background color, and the white text of the labels. The final customization is the highlighting of the cells. Only the schedule item cells can be highlighted to signify selection. The visualization for a selected cell is the change of the border color to blue.

Acquiring Content

Display Content Window

The purpose of the Display Content Window is to allow the user to view, and organize their content. The two main user tasks are, adding video content to their library, and creating/updating channels from their video library. The Display Content Window is accessed through a menu item in the Window menu.

User Interface

The interface for the Display Content Window is split into two different views. The views are laid out in a master-detail format. On the left hand side of the window is the source list. The source list serves as the master view that facilitates switching the content in the detail view. The source list contains two types of items, the library items that switch to showing the Video Items in a specific category (currently the only category is movies), and the channel items that switch to showing specific channels. On the right hand side of the view is the detail view. The detail view shows a collection of Video Items based on what is selected in the source view. For example if a specific channel is selected, the Video Items in that channel will be displayed in the detail view. The user interface also includes various buttons for adding/deleting Video Items and Channels.

Adding Video Items

Video Items are added to the user’s library using the Add Movie button. The Add Movie button causes an open panel to appear from the top of the window. Inside this panel is
displayed the user’s file system, which allows the user to choose content of the appropriate file types (see Table `<TODO>`). Once the user chooses a file to add the application automatically loads the information for the file, and then the file appears in the users library.

<table>
<thead>
<tr>
<th>Supported Video File Types</th>
<th>m4v, mp4, avi, mkv</th>
</tr>
</thead>
</table>

Table 4: Supported Video File Types

**Adding Channels**

The user can add Channels by selecting the add channel button in lower left hand corner of the window. Once the button is clicked a new Channel will appear in the source view, with the label 'untitled'. Selecting the source view item and pressing enter can change the channel label. To add Video Items to the Channel, items can be dragged from the library to the Channel’s source view item. Once a new item is added to the channel the Channel Switching View will automatically be updated to display correct information.

**Future Implementation**

Due to time constraints some of the project components from the initial design were not completed. The components have been left in the design section for future reference when their implementation beings. The not implemented components include, rich video metadata, automatic channel creation, and automatic metadata fetching.

**Testing and Validation**

Following is the testing status of the engineering requirements that were set up when the project was initially designed.

<table>
<thead>
<tr>
<th>Engineering Requirements</th>
<th>Testing Method</th>
<th>Result</th>
</tr>
</thead>
</table>
| Channel switching takes less than 2 second | • Switch to adjacent video  
• Switch to random video  
• Change channels with more than 10 channels  
• Change channels from keyboard, mouse, and remote | Pass  
Even with a large number of channels to switch between, the switching is still under 2 seconds. |
| Video controls disappear after 3 seconds of inactivity | • Test disappearing with no activity  
• Test staying visible with activity from any | Pass  
The controls only disappear when there is no user input activity. |
<table>
<thead>
<tr>
<th>Activity</th>
<th>Conducted Tests</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users should be able to add content in less than 1 minute.</td>
<td>• Have users try to add movies</td>
<td>Pass, Four users were tested, all successfully added movies.</td>
</tr>
<tr>
<td>Metadata will be fetched when a piece of content is added.</td>
<td>NA (feature not implemented)</td>
<td>NA (feature not implemented)</td>
</tr>
<tr>
<td>Schedules will be asynchronously created once timing runs out.</td>
<td>• Open application after time ranges have run out and make sure a video plays</td>
<td>Pass, Video plays correctly when the application opens after the time range, when the last video in the schedule ends, and when the next schedule is skipped to.</td>
</tr>
<tr>
<td>• Watch the last video in the schedule and make the first from the new schedule plays</td>
<td>• Skip to movie in the new schedule</td>
<td></td>
</tr>
<tr>
<td>Channels can be created manually</td>
<td>• Create channel by manually adding video content to it</td>
<td>Pass, Channels can be created and update manually using drag and drop.</td>
</tr>
<tr>
<td>• Test drag and drop video adding</td>
<td>• Have users try to add movies</td>
<td>Pass, Four users were tested, all successfully created channel.</td>
</tr>
<tr>
<td>Manually created channels can be created within less than 1 minute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channels can be created automatically based on metadata fields</td>
<td>NA (feature not implemented)</td>
<td>NA (feature not implemented)</td>
</tr>
<tr>
<td>The system will use no more than 1GB of memory</td>
<td>• Test memory footprint with more than 10 channels using the Instruments profiling tool</td>
<td>Pass, Maximum memory usage was less than 40MB</td>
</tr>
</tbody>
</table>

### Table 5: Testing Results
Conclusion

Many users have large movie libraries that contain the content they love. They acquired such content to be able to enjoy it many times over. The goal of this project is to allow each user to easily enjoy all their content without the overhead of searching through a giant list, and without having to commit to watching the entire piece. By automatically creating continuously playing channels the user will be able to switch it on and just enjoy.