<https://github.com/ForeEyes/PodcastFeed>

<https://github.com/pureooze/PodcastFeed>

<https://github.com/holmejmcm/PodcastFeed>

# Project scope & requirements

Using Qt to develop a program that allows the user to keep up with their favorite podcasts. The user enters a link to the iTunes/RSS podcast page. The program will remember the link, and fetch the xml data for the podcast into the users Documents directory. The user interface will have three panes below an audio player. The right most pane is context sensitive and will change based on what the user selects in the other two panes.

The left most pane will show the following (all read from the XML file):

* Podcast name
* Podcast artwork/logo
* Explicit/Not-Explicit

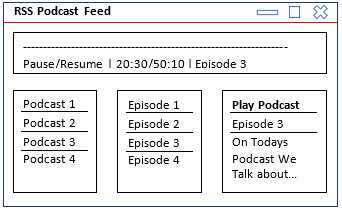
When a user left clicks the podcast name, the application will populate the middle pane with the list of episodes for the selected podcast. Additionally, the third pane will display additional information about the podcast that the user may find of use (name, author, number of episodes and description). The middle pane will show all the episodes of a podcast the user has selected in the left pane. It will also highlight any episodes the user has not already listened to.

If the user clicks on an episode in the middle pane, the right pane is populated with information about the specific episode (podcast name, episode name/number, episode description, episode runtime, episode partly listened to/fully listened to). If there are any links in the description, the user will be able to click them and the application will open them using the systems default web browser.

The media player will have standard play/pause buttons, a label showing the current playing time of the episode. The user will be able to seek the playing of the file forward and backwards using a slider widget. There will also be two buttons for skipping forward and backwards by 15 seconds.

# What did you do to achieve these requirements

To achieve these requirements, we first tried to visualize our end goal as a diagram (below):



This was helpful for use because it allowed us to design the UI in Qt to match our mockup without having to think about where certain features belong on the fly. Once we had an idea of what our GUI will look like we began to carefully define what each portion of the window would be responsible for displaying. For example, we decided that the podcast and episode panes only show content relevant to their respective name but the description pane on the right-hand side is context sensitive to what is selected in the other panes.

Once we had a clear definition of what each portion of the program would do, we decided to split up our work so that each group member would be responsible for developing parts of the program. We kept track of the progress of each individual feature using the issue tracking functionality on GitHub. As we worked on our individual components we came across useful documentation and tutorials that we could share on GitHub. By using this shared platform, we could rapidly develop and test features for our program.

In addition to our communications on GitHub, we held weekly meetings to discuss our progress on our project and how to best address any problems that may come up during development. It also gave us an opportunity to discuss how close we were to completing requirements that we stated in our project proposal. By taking the time to do these meetings we could focus our attention on features that were mandatory for the application before spending time on features that were less important.

# 3rd Party Library Usage

This project required the use of a variety of 3rd party classes that were all provided by the Qt framework. We chose to use Qt because it is a very popular and well supported C++ GUI framework that has been used to develop many popular programs like the Adobe Photoshop [1]. Qt also has support for streamlined multiplatform development because code can be written once on any platform but will compile and run under many other operating systems. For example, the code can be written on a Windows machine but can then be compiled for Mac and Linux systems without having to alter any of the code.

We also made use of several classes from the Qt framework and have listed the major ones with their respective use in the program below:

**QInputDialog:** Used to make a popup dialog appear on screen for a user to enter data that the program uses to remove a podcast. This class provides the ability to have a dialog open with an already configured “OK” and “Cancel” buttons. We chose to use this because it provided easy creation and handling on dialogs without having to manually implement every dialog for the program.

**QtNetwork:** Used to create an HTTP request to a webserver on the internet. This is used to download RSS files from the iTunes website. This is also used to store the reply from an HTTP request created by an instance of a QNetworkAccessManager object. This object allows waiting for the full request to be downloaded before continuing the process which is a useful feature as it prevents race conditions with missing portions of the downloaded files.

**QUrl:** Used to store URLs, this class can be used by several other classes to determine the path (URL) to a server on the internet. The function episodeFile() is an example of a function that returns an object of this type. For that function, the program will parse an XML file on the system, determine the URL to a specific podcast episode and return it for use in the parent function. This proved to be useful as we could use this QUrl object to load, track and unload episodes based on what widgets the user clicked.

**QMediaPlayer:** Used to play, pause, and stop media playback. This class is provided a URL to a media file and then the appropriate method is called to manage the play state and seeking. Since this class handles the parsing of audio files that are provided to it we could take advantage of this abstraction by using generic, reusable functions in our code.

**QBuffer:** Used to implement a custom buffer for the player. This is useful as on certain platforms where the OS seems to bottleneck the download speeds. We found this class useful to fully download the episode before starting to stream as this feature cannot be implemented with the basic QMediaPlayer class.

**QJsonDocument:** Used to parse the *podcasts.json* file to load the podcast information the user has already provided in a previous session. We use this class to handle the JSON key-value pair structure to quickly pull out information as needed.

Qt also provided a Signal & Slot construct to allow us to implement a chain of events based on certain criteria.

The following code is a sample of how we made use of various classes that are provided by the Qt framework to write an XML file to disk:

void MainWindow::storeXmlFile(QString podcastName, QByteArray rawReply){

//Podcast xml file path

QString podcastFile = xmlFolder + "/" + podcastName + ".xml";

QFile xmlfile(podcastFile);

//Open the file and write the data to it

if (xmlfile.*open*(QFile::WriteOnly) )

{

xmlfile.write(rawReply);

} else {

ui->statusBar->showMessage("Error Saving xml file", 3000);

}

//close file

xmlfile.*close*();

}

The above code is a method that is part of the MainWindow class, which is defined as the class which represents the program window. The QFile class is used to write to the disk by creating an instance *xmlfile* which is given the path stored in the QString podcastFile. We also use members of the UI and statusBar objects to display messages to the user.

# Challenges Faced

We came across several challenges when working on this project ranging from issues with the media player, downloading content from the internet, dealing with multiplatform development and with our software development process itself.

For managing our code and issues we made use of GitHub as our issue tracking and version control host. We chose to use GitHub because it allowed us to easily share code with each other without having to deal with the problems that emerge with using email to manage multiple versions of code. We also used the git utility to develop individual features in separate branches, allowing us to independently troubleshoot bugs and easily combine our code with the main repository. These separate branches also allowed us to easily drop certain portions of code if we decided to change how we wanted our program to work.

In addition to the code management issues we had several technical issues which are detailed on the issues section of our GitHub repository [2]. For example, a problem we ran into was with the QMediaPlayer class using a bottlenecked buffer on machines that run Windows. To resolve this issue, we used GitHub to share our thoughts and possible solutions about the bug. Eventually we could make use of some of the Qt documentation for the QBuffer class which allowed us to implement a custom buffer to fully load the media content before playing it for the user.

# Possible future additions or enhancements

* Downloading podcasts for offline playback
* Remembering the exact position
* Closing from taskbar
* Ffmpeg usage for improved playback support
* Cross platform executable testing