

Nikita Borodins

2328237

**BSc (Hons) Software Development**

**Programming Fundamentals (2023 MOD008904 TRI1 F01ARP)**

010-1 Component Programming Project

Glen Mason

(Anglia Ruskin University, n.d.)

Contents

[Requirement specification 2](#_Toc152595840)

[Meeting requirements 2](#_Toc152595841)

[UI 2](#_Toc152595842)

[Functionality 2](#_Toc152595843)

[Listeners 2](#_Toc152595844)

[TCP 3](#_Toc152595845)

[InboundConnectionListener 3](#_Toc152595846)

[TcpSessionListener 3](#_Toc152595847)

[UDP 3](#_Toc152595848)

[IP 3](#_Toc152595849)

[UdpListener 3](#_Toc152595850)

[ListenerCollection 4](#_Toc152595851)

[Handling data 4](#_Toc152595852)

[Handler 4](#_Toc152595853)

[User Interface 4](#_Toc152595854)

[NavClass 4](#_Toc152595855)

[Settings 4](#_Toc152595856)

[Changing Colours 5](#_Toc152595857)

[Defining filters 5](#_Toc152595858)

[Changing listening interface 5](#_Toc152595859)

[Changing saving directory 6](#_Toc152595860)

[Main Menu 6](#_Toc152595861)

[Listener management 6](#_Toc152595862)

[Viewing all messages 7](#_Toc152595863)

[Testing 7](#_Toc152595864)

[What could have been done better? 7](#_Toc152595865)

[What went well 8](#_Toc152595866)

[Lessons learned 9](#_Toc152595867)

[Things I did not know 9](#_Toc152595868)

[Things I improved my knowledge in 9](#_Toc152595869)

[Usage guide 9](#_Toc152595870)

# Requirement specification

According to the requirements specified, a Syslog program is to be produced with a few fundamental requirements:

Support for:

* [TCP(minimum of 5 concurrent connections must be able to be supported at a time).](#_TCP)
* [UDP](#_UDP).
* IPv4.
* IPv6.

The ability for the user to change:

* Colors for different severity levels.
* Filtering conditions.
* (implied) Directory to which all exports are saved.
* Listening NIC for all listeners (Current connections are implied to remain as they are).
* Protocol that is used when listening.
* Ports to listen on.

Some program specific requirements have also been defined:

* Code standards: ParkAir C# Standards –

 .

* Use of C# programming language.
* [Use a CLI.](#_UI)
* [Repeating large blocks of logic is disallowed(memory efficiency)](#_NavClass)

Resources used:

* Microsoft windows 10
* Visual Studio 2022
* .NET 6
* C# version 11
* GitHub

# Meeting requirements

## UI

The UI has a lot of components and us built on a CLI

## Functionality

### Listeners

There are two Transport protocols that are required to be used : UDP and TCP one prioritized integrity and uses sessions, and the other is Connectionless, but does not guarantee integrity.

Both listeners inherit from the Listener class



This makes them both covariant to Listener, and therefore expose their intended functionality through the method that Listener exposes. This allows them all to get checked for messages from one connection. This functionality could have been achieved without inheritance, but looks cleaner.

checkForMessages() is a method that they bot inherit, and as per the name, it is intended to check whether there are new messages inbound.

ChangeIp() was not in the Listener base class, because ChangeIp is implemented by InboundConnectionlistener and not session specific listeners, this would mean that InboundConnectionListener would need to implement redundant methods

### TCP

TCP is implemented using two classes, one to handle incoming connection requests, and another to handle each individual connection

### InboundConnectionListener

– handles incoming connections



### TcpSessionListener

– handles each connection. Something noteworthy about it is that it disposes itself when a connection ends, which is not a requirement but was implied



## UDP

UDP was simpler to implement than TCP. One class was used to implement UDP

## IP

IpAddress supports both IPv6 and IPv4, so there shouldn’t be issues with using IPv6 or IPv4 using this program. Errors may occur if the network isn’t configured to use IPv6

### UdpListener



## ListenerCollection

ListenerCollection is meant to handle all of the listeners in this application. It uses one thread to check for inbound connections and one thread to handle all new messages. It takes a handler delegate as one of its parameters and invokes it with the message as a parameter for each message. It uses a model to associate a IpEndPoint with each message as the sender.



# Handling data

## Handler

There is a single class that handles all of the inbound messages.

It uses a ConcurrentQueue to queue up all new messages that are to be handled. It puts the message into a specialized model that is designed to separate all individual attributes of a message so the user can use filtering on them.



# User Interface

The user interface is primarily oriented in a view-like pattern, you select an element and get sent to another view/

An important feature of most such views, is that they use the singleton pattern, meaning that you cannot have more than one active instance at a time. This is done to reduce the complexity and reduce the space for errors.

most UI element inherit from [NavClass](#_NavClass_1)

## NavClass

NavClass uses the pattern of most navigation pages using the same logic, since they often act as links/selectors for other more specifically purposed components that alter the behaviour of the program. This reduces the repetition of logic.



## Settings

There I a settings view, which allows the user to [define filters](#_Defining_filters), [change severity colours](#_Changing_Colours), [Change listening network interface](#_Changing_listening_interface) and [change the directory to which logs are saved](#_Changing_saving_directory)



The settings menu itself, is a “link” UI element, which is meant to expose other functionality as opposed to providing Functionality itself. This is except a small caveat in the [directory changing function](#_Changing_Colours)



### Changing Colours

Changing colours is a simple UI element. it lets a user select what colour they would like to be associated with what severity level.



### Defining filters

Defining filters is a fairly complicated mechanic that ahs to be implemented. I had to design a model hat would increase interactivity with the data while keeping the system secure and keep the data accessible considering the filtering constraints that I have set up to keep the system safe.

The Model is:



The definition f the filter itself is contained in this view:



There is also supposed to be a separate user guide, that will be given at a later point, but to summarize, you can reference the objects using the characters that satisfy the regex “^[a-zA-Z0-9.<>= |&!?:+-]+$“ I restricted the use of brackets and quotes because they can allow the user to invoke methods they aren’t supposed to. Using literals such as 12.12.3.23 will result in them being parsed in as a IP address. If they are something such as 555.555.555.555, then a runtime error will be thrown during the test execution and the user will be prompted to enter an alternative input. All compilation errors and runtime errors are outputted to the user.

In general, the filtering pattern uses C# syntax.

### Changing listening interface

Changing the listening interface sets all active UdpListener and inboundConnectionListener instances to listen on that endpoint. Existing connections on TCP will persist until the sender disconnects.

### Changing saving directory

This is one of the few elements that are a method as opposed to an entire class of itself. This increases coupling of the program, however the method is so simple that I thought it would be simpler to make it into a method also it would likely be more inefficient, since if it inherited from NavClass, it would need to implement all of its methods, which is unnecessary.

All this method does, is get a string and make sure it’s a valid directory and format it to be compatible with all of the writing that other classes do.

To see implementation, please navigate to the GetDirectory() Method in SettingsView



## Main Menu

The main menu itself, is another “link” class that is meant to give the user a selection between few functionalities accessible by them.



### Listener management

Listener management allows the user to create new Listener instances on specific ports of the default NIC, or Remove them

Here is the “link” class that allows the user to select more specific functionality



#### Creating Listeners

Creating listeners is simple from the user’s standpoint, just press create, and you will be prompted on the port you want to listen on, you can enter it and it will get validated. Contrary to the requirements, there is not default port that is usually listened on. The UI does not inherently allow errors, so there is no “defaulting” behaviour.



#### Removing Listeners

Removing listeners is also simple, since all the user needs to do I select what port they want to remove the listener on(they are prompted with a menu) or they can exit.

When removing a InboundSessionListener , all TcpSessionListener instances associated on that port will also be removed

# Viewing all messages

The ListeningView implementation has a lot of divergence from NavClass, which is not too important, however, its concrete version implements methods that aren’t implemented by NavClass, so they are not to be used interchangeably if in question.

IMPORTANT:

Pressing Esc escapes the view, pressing Tab, saves all messages and then leaves



# Testing

Through development, testing had to be done to ensure that there are no uncaught/unforeseen errors. A log of such tests has been produced.



# What could have been done better?

Overall, I think that the program meets most of the requirements specified.

What I have not done is :

* Implement secure Syslog -I could have used TLS to encrypt data in transmission and made the transmission more secure.
* Efficiency – the program is not overly efficient, due to the fact that it uses a lot of IO bound operations in places that could be relegated to other places such as another worker thread, or async/await ( I don’t think this would be as good of a solution since this program does not use EDP to the extent that would make it actually beneficial. A good example of the inefficiencies of this program would be A computer screen shot of a program

  Description automatically generated

In here, I could have allocated the check if the connection is still active to another thread, and the IO bound task would still work, just on another thread. I could have used a similar structure to that of the [Handler](#_Handler)

* Coupling – to a good extent, this program does utilize loose coupling, however some things where implemented in a way that specializes them in a very specific way(s).
* Such as the [SettingsView](#_Settings) having a method inside it to get user input. This specializes this class in a way that makes it unusable in a lot of other situations. Luckily, inheritance from NavClass generalises the functions of the UI, and not too much of a loss had been made from that
* Locking – I have used the same lock in places, where another really should have been used, for example using the same lock for two collections in [ListenerCollection](#_ListenerCollection) .

# What went well

* Overall, this program runs fine, and doesn’t seems to have any errors.
* The dynamic compilation of the filter was a very impressive feature of this program. I thought it was going to be the most difficult element, but was definitely not that difficult.
* Working with UDP and TCP now, will allow me to implement them in more efficient manners in the future.
* Parallelism does not seem to have any issues, despite there being some potential improvements I could have made.

# Lessons learned

## Things I learned

* What an atomic operation is.
* Dynamic compiling

## Things I improved my knowledge in

* Loosely coupled applications
* UDP
* TCP
* IP
* Syslog protocol

# Usage guide



# References

Anglia Ruskin University, n.d. *University courses at ARU | Anglia Ruskin University - ARU.* [Online]   
Available at: https://www.aru.ac.uk/  
[Accessed 05 12 2023].