|  |
| --- |
| Roster Utility |
| COMP4 Project |
| James Knatt, Godalming College |



Centre no.: 64395

Candidate no.: 8158

Contents

[Investigation and Analysis 5](#_Toc383429348)

[The Organization 5](#_Toc383429349)

[Summary of research methods 5](#_Toc383429350)

[‘What is a roster?’ – a short summary 5](#_Toc383429351)

[The Interview 6](#_Toc383429352)

[Summary of Interview Responses 8](#_Toc383429353)

[Document collection 9](#_Toc383429354)

[Explanation of key roster elements 10](#_Toc383429355)

[Observation 11](#_Toc383429356)

[The questionnaire 13](#_Toc383429357)

[Summary of Questionnaire Responses (with graphs) 14](#_Toc383429358)

[Conclusion of research into employee interface 15](#_Toc383429359)

[Research into hardware 15](#_Toc383429360)

[Problem summary (after research methods completed) 16](#_Toc383429361)

[Description of Existing System (inc Flow Chart) 16](#_Toc383429362)

[Systems outline chart (current system) 18](#_Toc383429363)

[Users in the current system 18](#_Toc383429364)

[Data Sources and Destinations 18](#_Toc383429365)

[Data Flow Diagram (existing system) 19](#_Toc383429366)

[Entity Relationship Diagrams 19](#_Toc383429367)

[Data volumes 19](#_Toc383429368)

[Objectives for the Proposed System 20](#_Toc383429369)

[Potential Solutions 21](#_Toc383429370)

[Chosen Solution 24](#_Toc383429371)

[Flow Diagram of Proposed System 25](#_Toc383429372)

[Evaluation of Analysis 26](#_Toc383429373)

[Design 27](#_Toc383429374)

[Systems outline chart 27](#_Toc383429375)

[Full system top down design 27](#_Toc383429376)

[Entity-relationship diagrams 27](#_Toc383429377)

[Input design summary 29](#_Toc383429378)

[Output design summary 29](#_Toc383429379)

[Process design 29](#_Toc383429380)

[Identification of suitable algorithms for data transformation 32](#_Toc383429381)

[Example pseudocode of a suitable algorithm 33](#_Toc383429382)

[Definition of data requirements (Design Data Dictionary) 34](#_Toc383429383)

[Example validation required 35](#_Toc383429384)

[Identification of storage media 36](#_Toc383429385)

[Hardware specification 36](#_Toc383429386)

[User interface design (rationale) 36](#_Toc383429387)

[UI sample of planned data capture and entry designs 37](#_Toc383429388)

[Sample of planned valid output designs 39](#_Toc383429389)

[System security and data integrity 39](#_Toc383429390)

[System testing – outline test plan 40](#_Toc383429391)

[Alpha testing design (Input/Flow of control/File system testing) 40](#_Toc383429392)

[Alpha testing of algorithm and decision processes 42](#_Toc383429393)

[Beta testing against specification 42](#_Toc383429394)

[Testing 45](#_Toc383429395)

[Outline test plan 45](#_Toc383429396)

[Testing datasets 45](#_Toc383429397)

[Input testing table 46](#_Toc383429398)

[Flow of control testing table 47](#_Toc383429399)

[Iteration and decision testing 48](#_Toc383429400)

[System testing 49](#_Toc383429401)

[System testing table 49](#_Toc383429402)

[System testing against specification 50](#_Toc383429403)

[Maintenance 51](#_Toc383429404)

[System overview 51](#_Toc383429405)

[Measures to ensure security of data 52](#_Toc383429406)

[Summary of package(s) used 52](#_Toc383429407)

[Evidence of corrective, adaptive and perfective maintenance 52](#_Toc383429408)

[Structural overview of the forms 57](#_Toc383429409)

[Sample screenshots of forms 59](#_Toc383429410)

[Structure diagram of the forms – top down design 61](#_Toc383429411)

[List of all the variables used in the system: name, access, type and description 63](#_Toc383429412)

[List of all the procedures/functions used in the system 67](#_Toc383429413)

[Algorithm overview 72](#_Toc383429414)

[Pseudocode of the created algorithm (modified from Design section) 72](#_Toc383429415)

[Coded functions/subroutines for the created algorithm (from above pseudocode) 72](#_Toc383429416)

[Algorithm design 75](#_Toc383429417)

[Specific flowcharts for roster processing: calculatePenalty 78](#_Toc383429418)

[Specific flowcharts for roster processing: findAllocationLowestPenalty 80](#_Toc383429419)

[Annotated program listing/code 81](#_Toc383429420)

[System settings/Configuration 81](#_Toc383429421)

[Research into changeover methods 82](#_Toc383429422)

[Changeover method for the roster system 83](#_Toc383429423)

[User Manual 85](#_Toc383429424)

[Appraisal 86](#_Toc383429425)

[Comparison of solution to original objectives 86](#_Toc383429426)

[Deadlines 89](#_Toc383429427)

[User feedback on the roster solution 89](#_Toc383429428)

[Analysis of user feedback 90](#_Toc383429429)

[Extensions 91](#_Toc383429430)

[Appendices 93](#_Toc383429431)

[Appendix A – document analysis 94](#_Toc383429432)

[Document 1 – TRN Proposal 130513.docx 94](#_Toc383429433)

[Document 2 - TRN Team A v01 280513.xlsx 96](#_Toc383429434)

[Appendix B – program listing 97](#_Toc383429435)

[createCustomShifts1.vb 97](#_Toc383429436)

[createRoster1.vb 99](#_Toc383429437)

[createCustomShifts2.vb 101](#_Toc383429438)

[createRoster3.vb 102](#_Toc383429439)

[globalVariables.vb 111](#_Toc383429440)

[mainForm.vb 111](#_Toc383429441)

[rosterGridView.vb 115](#_Toc383429442)

[Appendix C – Test data 117](#_Toc383429443)

[Section 7 – hand traced algorithm design 143](#_Toc383429444)

[Roster Utility v0.1 User Guide 1](#_Toc383429445)

[Welcome to Roster Utility! 2](#_Toc383429446)

[Introduction to system 2](#_Toc383429447)

[Minimum technical specifications: 2](#_Toc383429448)

[Installation procedure 3](#_Toc383429449)

[Tasks within the system 7](#_Toc383429450)

[Creating and viewing a new roster 7](#_Toc383429451)

[More complicated rosters – custom shift patterns 11](#_Toc383429452)

[Viewing rosters 13](#_Toc383429453)

[Clearing rosters 15](#_Toc383429454)

[Viewing help 16](#_Toc383429455)

[View the Knatt Consulting website 16](#_Toc383429456)

[To quit the Utility 17](#_Toc383429457)

[Troubleshooting 18](#_Toc383429458)

[How to uninstall the program 19](#_Toc383429459)

[Backup Procedures 20](#_Toc383429460)

[Legal 20](#_Toc383429461)

175

[Bibliography 159](#_Toc383429462)

# Investigation and Analysis

### The Organization

Knatt Consulting is a local small business which provides a range of business consulting services to the aviation sector. The company is based in Plaistow, West Sussex but has had clients from London Heathrow to Madeira Airport, and more recently is providing management advice in Dublin. The Managing Director is Chris Knatt, who is likely to be my primary user.

Location: Telephone:

The Oaks 07502 018202

Dunsfold Road

Plaistow Email:

RH14 0PW chris@knattconsulting.com

### Summary of research methods

Interview

An interview is the best type of data collection for a single user – it provides in-depth knowledge as long as the interview subject is willing to discuss the organization’s requirements in front of the interviewer. An interview requires substantial effort so a project is usually limited to a certain number of interviews. Data received from interviews is harder to collate, but answers can be more substantial and the user isn’t constrained if they want to extend their views (unlike with questionnaires).

Questionnaire

A questionnaire allows data collection from a large number of users. It’s easy to interpret the data which is received. However, questionnaires can provoke short, less detailed responses as interviewees may not expand on their idea, resulting in a shortage of useful data.

Record inspection

Record inspection is time consuming and can be unreliable. It involves looking at all of the existing documentation that exists for the current system, and analysing it in an attempt to see how the existing system has functioned in the past. It may be used in conjunction with other data collection methods but should not be relied upon for the sole source of data.

Observation

Observation requires taking a ‘look’ into how the company is operating. This can be useful to resolve conflicting views from other data collection methods but it will only give a limited ‘look’ into the company’s operation. It also takes a lot of time and requires a written report or summary to be created.

### ‘What is a roster?’ – a short summary

A roster is defined as ‘a list or plan showing turns of duty or leave for individuals or groups in an organization’. However, the term ‘roster’ is also used to describe the process of creating a roster – assignment of shifts to individual employees. In this project, both meanings of the term will be used. In short, employees and managers are assigned shifts (‘rostering’), and these shifts make up a pattern known as a ‘roster’, which contains each shift, and is usually stored as a document so it can be referred to.

### The Interview

I conducted an interview with Chris Knatt, the Managing Director of Knatt Consulting (who provide the roster proposals to the clients). This was done in order to gain an insight into the following areas:

* The current system and his part in the running of the system, including specifics for at least one recent client project
* The problems with the current system
* His views on the required features of the new system
* Which platform the system is intended to be developed for

The interview method was used to provide an in-depth response from the primary user’s viewpoint, allowing me to find out key details in an unrestricted manner. This method provides a large amount of data and is instrumental in the problem analysis and definition.

Below is a transcription of the interview, which took place on the 21/08/13. The questions I prepared are shown in *italics*.

*Describe the current system including your role in the operation.*

Various clients use the services of Knatt Consulting to create and propose amended roster patterns when the needs of their business change. At these times, Knatt Consulting uses the existing roster pattern along with the requirements for the new roster to create a new proposal.

For example, in the case of the rail operator project, the company employs a number of different types of staff to run their operation from drivers of trains to customer services representatives. All of these staff are shift-based workers who need to be available at times to suit the needs of the train operating timetable. This runs 7 days a week from 5 o’clock in the morning until nearly midnight each day.

When the train timetable is changed (eg for winter or summer seasons) or the needs of the business change in some way, the rosters worked by the staff need to be amended to suit.

*What are the problems with the current rostering system? (eg significant errors)*

When developing potential rostering solutions, there is a need to ensure that a sufficient number of staff are available at the right times to cover all of the work required. There is also a need to ensure that the proposed work pattern requires staff to work no more than their contracted number of hours per week. Other requirements exist such as a minimum time between finishing a late shift and starting on an early shift on the following day. All of these checks are currently done in bespoke Excel spreadsheets and are therefore time consuming and prone to calculation errors.

There are also additional issues with subcontracting the rostering: it can take up to seven or even ten days for the expert employed to create a new roster or amend the existing one, and this then has to be returned to Knatt Consulting who ensure that it meets the client’s requirements. This means that the current system is inefficient and requires a significant amount of time to create any new proposals. It is also quite likely that the client’s requirements may be misinterpreted and so time will be wasted producing solutions for requirements that do not actually apply.

*What is the most important part of a successful rostering program?*

Roster software needs to quickly develop roster patterns and accurately validate them against a given set of rules. A good example of this is within the rail operating company: as some of the staff roles are deemed to be safety critical by the rail regulator, not having the right number of staff in the right place at the right time could ultimately mean that the operation cannot run. Therefore, rules are in place in the current system to ensure that this does not happen.

*With a specific roster, how many employees need to be accounted for? How often is a typical roster renewed?*

Clients’ employees are often split into smaller groups. For example, the rail company employs a total of 435 staff working on a variety of different rosters relating to their particular duties. The main groups are drivers (82 staff) and Customer Services Representatives (176 staff). These two groups change rosters at least twice a year in line with the train timetable.

*How are the shifts set up?*

Roster solutions are either developed by manually modifying existing patterns or creating complete new solutions. The number of work days and rest days and other characteristics of the roster should be configurable parameters as they will vary from group to group, and obviously between different clients.

*What platform should the new system be developed for?*

Wherever possible, the new system should be developed for Windows based PCs running standard desktop software without a great need to purchase bespoke applications. The bespoke element would ideally involve a single program used to produce the rosters, but if so it should export the rosters in a standard file format readable by other applications. The machine used by Knatt Consulting is running Windows 7 and has a reasonable specification so it should be able to handle a new system without the need to purchase more hardware.

*What would be the most suitable method for displaying the roster to the employees, if the solution was extended to allow this?*

Some form of web-based application would be preferred with the potential to develop a mobile solution too. As a business, Knatt Consulting has a website and the web server could be used to display this (if this was of any use).

*How important is validation of a created roster?*

It is far more important to validate a created roster than simply introduce a randomly generated one which may not meet the client’s requirements. To this end, the new system must use specific rules to create new rosters which adhere to the rules entered. It might also be quite hard to create a new roster without any rules to start from!

*If a solution was to be produced to solve this problem, when would the specific deadline be for this to be completed?*

The project would have a reasonable time frame – a solution is likely to need to be completed by January or February 2014, since several larger projects begin in March, and they will require the system to function, and the swapping over to be completed.

### Summary of Interview Responses

From the analysis of the interview, Mr Knatt has been able to clearly express his requirements for the new system. The new system must replace the current subcontracting of the roster – this is currently done ‘manually’ and therefore has potential to be much more efficient. This is well-suited to a bespoke program to be run on a standalone computer (in Knatt Consulting’s office). If the system’s functionality was to be extended, a web-based interface could be created to display the roster to the employees, with a possibility of facilitating requests for holidays. However, additional research would need to be done into this area, and besides, roster display is clearly less valued by Mr Knatt than efficient and accurate roster creation.

From the interview it would seem that the majority of the system’s data processing takes place in the actual creation of the roster – currently done by the expert employed by Knatt Consulting. It is important that the rosters the program creates fit given constraints, as some shift work cannot run without a set number of people present. Since the program is to create rosters that are suitable for different clients (within set limitations) it is important that I do not simply focus on one problem within the system (the train company, for example) and instead that I create a program that is able to work with a small range of data so that it is applicable to more than one client. Even more essential is the capability of the new system to *validate* the created rosters, so they meet the rules they were created with. This also needs to take place when a new roster is created.

### Document collection

Mr Knatt was able to provide me with some of the documents which play a part in the existing system. These consisted of an existing proposal for a specific client company (*TRN Proposal 130513.docx*) and an existing roster for this company (*TRN Team A v01 280513.xlsx*). Below are document specification sheets that I compiled for these two documents. The documents are attached in Appendix A.

TRN Proposal 130513.docx

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Volumetrics** | | | | | | | | | | | |
| Document description | | | System | | | | Document | | Name | | Sheet |
| Proposal  (TRN Proposal 130513.docx) | | | Rostering system | | | | 1 | | James Knatt | | 1-3 of 3 |
| Stationery ref. | | | Size | | | | Number of parts | | Method of preparation | | |
|  | | | A4 | | | | 2 | | Typed | | |
| Filing sequence | | | | | Medium | | | | Prepared by | | |
| Chronological | | | | | Digital/ring binder | | | | Consultant | | |
| Frequency of preparation | | | | | Retention period | | | | Location of file | | |
| 1x per new roster proposal | | | | | >1 year | | | | Consultant’s office | | |
| Volume | Minimum | | | Maximum | | Av/Abs | | Growth rate/fluctuations | | | |
| 0 | | | 1 | | 1 | | 1-4 created per year.  Fluctuates seasonally – when new roster proposal required by client company. | | | |
| Users/receipts | | Purpose | | | | | | | | Frequency of use | |
| 1 – Client company  2 - Consultant | | Review proposal  Keep for records | | | | | | | | X1  X1 | |
| **Data Dictionary** | | | | | | | | | | | |
| Ref | Name | Data Type | | | Length | | | Occurrence | | Source of data | |
| 1 | Logo | OLE | | | 1 | | | 1 | | Consultant | |
| 2 | Address | X | | |  | | | 1 | | Consultant | |
| 3 | Body | X | | | 626 | | | 1 | | Consultant | |
| 4 | Footer | X | | | 15 | | | 1x per page | | Consultant | |
| 5 | Signature | OLE | | | 1 | | | 1 | | Consultant | |
|  |  |  | | |  | | |  | |  | |
|  |  |  | | |  | | |  | |  | |

TRN Team A v01 280513.docx

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Volumetrics** | | | | | | | | | | | | | | | | |
| Document description | | | | | | System | | | | Document | | | | Name | | Sheet |
| Roster  (TRN Team A v01 280513.xlsx) | | | | | | Rostering system | | | | 2 | | | | James Knatt | | 1 of 1 |
| Stationery ref. | | | | | | Size | | | | Number of parts | | | | Method of preparation | | |
|  | | | | | | A4 | | | | 3 | | | | Typed | | |
| Filing sequence | | | | | | | | | Medium | | | | | Prepared by | | |
| Chronological | | | | | | | | | Digital/ring binder | | | | | Consultant | | |
| Frequency of preparation | | | | | | | | | Retention period | | | | | Location of file | | |
| 1x per new roster proposal | | | | | | | | | >1 year | | | | | Consultant’s office | | |
| Volume | | Minimum | | | Maximum | | | Av/Abs | | | Growth rate/fluctuations | | | | | |
| 0 | | | 1 | | | 1 | | | 1-4 created per year – DIFFERENT versions.  Fluctuates seasonally – when new roster proposal required by client company. | | | | | |
| Users/receipts | | | | | | Purpose | | | | | | | | | Frequency of use | |
| 1 – Client company  2 - Consultant  3 – Rostering expert | | | | | | View completed proposal  Keep for records  Modify to best fit new client’s needs | | | | | | | | | X1  X1  X1 | |
| Data Dictionary | | | | | | | | | | | | | | | | |
| Ref | | | | Name | | Data Type | | | Length | | | | Occurrence | | Source of data | |
| 1 | | | | Title | | X | | | 4 | | | | 1 | | Consultant | |
| 2 | | | | Weekday | | X | | | 1 | | | | 7 | | Consultant | |
| 3 | | | | Line | | A | | | 1 | | | | 1 | | Consultant | |
| 4 | | | | Name | | X | | | 1 | | | | 1 | | Consultant | |
| 5 | | | | On | | X | | | 1 | | | | 7 | | Consultant | |
| 6 | | | | Off | | X | | | 1 | | | | 7 | | Consultant | |
| 7 | Dia | | X | 1 | | | 7 | | | | | Consultant | | |
| 8 | Hours | | X | 1 | | | 7 | | | | | Consultant | | |

### Explanation of key roster elements

Below is a section from the existing roster *TRN Team A v01 280513.docx* which is analysed above. I have labelled the key sections that rosters created by my solution might need to have.



Number of hours worked by an employee

Times when shifts start and end

Shift assignment (RD stands for rest day)

Listing of all employees in roster

### Observation

As an additional part of my investigation into the rostering problem, I decided to research into the approximate content of emails to and from the rostering expert over the period of creation of the new roster. This provided me with real-life examples of how a previous rostering problem (for a client company’s ‘Rev Centre’ shift work) was solved. As the emails are stored electronically, I was able to access the date and time that each message was received: this enabled me to work out the approximate time it takes for a new solution to be created.

I have summarised the important points contained in each email:

12 May 2013 – 09:20 [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com) > [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk)

* This email supplied problem-specific details of one group in one location in the roster.
* Mr Knatt also set the specialist two tasks: creation of a new roster for a specific line in this solution and modification of the existing roster to remove Saturday PM shifts.

12 May 2013 – 11:22 [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk) > [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com)

* In this email the specialist confirmed that an update on the rostering would be provided in the afternoon of the 12 May.

12 May 2013 – 22:09 [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk) > [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com)

* This email notified the details of the specialist’s attempt at creating the new roster, and how it was produced (condensing existing lines).
* It also notified Mr Knatt of the problems encountered while doing this, and the total number of hours spent on the solution.

13 May 2013 – 09:28 [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com) > [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk)

* In this email Mr Knatt confirmed specific details of the shifts to be edited.

14 May 2013 – 22:48 [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk) > [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com)

* Here the specialist contacted Mr Knatt, attaching two different roster solutions from the updated information about the shifts.

15 May 2013 – 08:41 [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com) > [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk)

* A meeting was organised at Heathrow Airport.
* Mr Knatt provided feedback on one of the potential solutions (splitting the group into two halves).

19 May 2013 – 22:26 [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk) > [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com)

* The specialist attached a draft summary report for the proposed roster system to this email.
* The meeting was confirmed for the 20 May.

21 May 2013 – 23:28 [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk) > [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com)

* The specialist attached the new roster, with reviewed/amended points as discussed at the meeting.
* A new ‘team day’ had also been added to the roster as requested by the client.

22 May 2013 – 09:07 [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com) > [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk)

* Mr Knatt thanked the specialist for the new roster and organised another meeting for the 31 May.

22 May 2013 – 20:56 [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk) > [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com)

* The specialist confirmed the meeting, attaching three different solutions for the project based on three different ways of aligning the shifts.

27 May 2013 – 16:23 [chris@knattconsulting.com](mailto:chris.knatt@knattconsulting.com) > [waq\_\_@hotmail.co.uk](mailto:waq__@hotmail.co.uk)

* Mr Knatt attached a summary of the Rev Centre work in the format that he sent it to the client (CL1 Rev Centre Rosters Draft Summary 240513.pdf)
* Some predictions were made for the client’s feedback on the new solution.
* The next project was discussed.

In total the new solution took around two weeks to produce (10 May – 27 May). This involved two meetings with the roster specialist and multiple emails as the client provided feedback on the solutions in progress. It is worth noting that solving the Rev Centre project would require a very complex program, and that it is likely to be beyond the planned capabilities of my system at the current time. However, this observation demonstrates the difficulties with subcontracting the work, and that the current manual system takes a significant amount of time for a new solution to be produced by hand. Through computerisation of the rostering and removal of the subcontraction, my system should reduce the time taken for Mr Knatt to create and validate new rosters.

### The questionnaire

In conjunction with the other methods of research that I conducted, I designed a questionnaire for a sample of the train company’s employees to gauge their views on a possible *extension* of the system to include the previously-discussed frontend, or employee interface. I chose a questionnaire method of research as I felt that interviewing multiple employees would be time-consuming and would not provide enough useful data to offset the effort that I would have to put into its collation. I designed the questionnaire to allow averages to be calculated and graphs to be plotted of the employees’ responses, allowing easier interpretation of the greater volume of data.

A replica of the questions used is shown here.

**Questionnaire**

This questionnaire is to give me more of an idea about what you would like to see from an extension of the planned system. The questions cover the problems with the current system, and what your needs are for the new system’s potential employee interface.

How satisfied are you with the current rostering system? (1-10) (Leading towards a simplification of the system to improve the ease of use)

*Not satisfied* 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10 □ *Very satisfied*

How accurate is the current rostering system? (1-10) (Benefits of computerisation of system)

*Not accurate* 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10 □ *Very accurate*

If the system was modified to include a form of timetable display, how important would it be to you to be able to **request** holiday more than annually? (1-10) (Specific task that the users may need to be able to do with the extended system)

*Not important* 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10 □ *Very important*

With this modification, how important would it be to you to be able to access the roster at **any time**? (1-10) (Again, a specific task for the extended system)

*Not important* 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10 □ *Very important*

**How** would you prefer to access your part of the roster? (Opinions on the best platform for the extended system’s viewing application)

Mobile application □ Web-based interface □ Bespoke program in the workplace □

Paper-based system □

Do you personally own a device that could access a **web-based interface**? (Staff members who own such a device are likely to be more computer-literate and would find the interface easier to use)

Yes □ No □

Have you had **previous experience** of a web-based rostering interface in the workplace? (eg MyRoster/RosterNet) (Again, relating to how easy the staff will find it to adapt to a new system)

Yes □ No □

### Summary of Questionnaire Responses (with graphs)

Here the questionnaires show that overall this sample of employees are reasonably satisfied with the rosters, with an average of 5.6/10. The current system in place seems to function adequately, and by removing the manual element this will not change significantly. However, an extension involving a simpler form of roster viewing would increase satisfaction and would therefore be welcomed.

It would also appear that the current system in place lacks a great deal of accuracy, averaging just 4/10. This is most likely to be due to the manual elements which introduce human error, and in particular the fact that the rosters are produced without using any sort of algorithm. The new system must therefore aim to be much more accurate, using an algorithm of some sort to produce rosters with no faults.

A number of employees at the rail operating company have to work irregular shifts, and therefore accessing one’s personal part of the roster at any time would be an important part of the potential employee interface. One employee in this sample would not be able to check his shifts at home and so placed less importance on access at any time: however, the roster needs to be available to view at all times in the workplace.

Almost all of the employees here are fairly computer literate, therefore a web-based solution for accessing the completed roster continues to seem like the best solution if I was to extend the program to allow this. A few suggested a mobile app would be better, but a web solution should be platform-independent and would work on the majority of smartphones as well as both PCs and Macs in the workplace and at home. Hosting the roster on a server and creating a HTML frontend seems sensible, and less costly and time consuming than creating a second piece of bespoke software dedicated to viewing the rosters.

Ninety percent of these employees own a device outside of the office which can access a web-based solution. This supplements the preceding question, supporting the idea of a web solution. The one employee who does not own a device in this sample can access a PC in the workplace to view and print off their shifts if necessary: this means that they are not hindered by this part of the system becoming computerised. Even if this proportion was extrapolated those employees could still access the system at work.

Some employees in this sample have previously experienced a web-based roster viewing solution; for others this would be their first experience. Since the majority of employees are fully computer literate, they should have no problems using a well-documented system to view the roster. Those employees who have used a system like this before can also be trained to help newly-employed workers start to use the system for themselves, for example.

### Conclusion of research into employee interface

The questionnaire I produced considered the suitability of an employee interface (where the shift workers can view and interact with their personal timetable). From these, it seems that an interface like this would be very well suited to the rostering problem should I wish to implement it – and it would greatly improve the end-user experience – but I have to consider the amount of additional time it will take for me to develop and implement. The primary user (Knatt Consulting) places far less importance on the employee interface and more on the creation of an accurate roster in a short amount of time, without a third party’s involvement. Therefore, I shall complete the bespoke program to do this before attempting to extend the project. It seems sensible that the employees would value fair and error-free shift organisation over simply having the ability to view their shifts in a slightly easier way.

|  |
| --- |
| **Primary user’s hardware (from research)** |
| Intel Core i5 (2.30GHz) |
| 4096MB RAM |
| 1333x768 display resolution |
| ~20GB available on HDD, access to several 4GB memory sticks for backup/transfer |
| USB2 ports for transfer |
| Broadband LAN connection |

### Research into hardware

During my investigation, it is important that I look at the hardware currently in place to determine whether it is suitable for my new system. Since upgrading hardware can be expensive and is time consuming, research must be done to determine if the creation of the new system is feasible on the current hardware. From research into the specification of his computer, Mr Knatt currently owns a mid-level PC running suitable components to run elements of my new system without the need to upgrade.

### Problem summary (after research methods completed)

During the course of the previous year, the company has been asked to provide shift rostering solutions for several different clients, including airport operators and a major London rail operating company. Currently, Knatt Consulting is subcontracting the detailed development of new and amended rosters as and when major changes are required (1-3 times per year). I feel that a computerized system would benefit the primary user (Mr Knatt) by allowing faster modifications to be made to the rosters, as well as removing the need to subcontract and thus saving the company time and money.

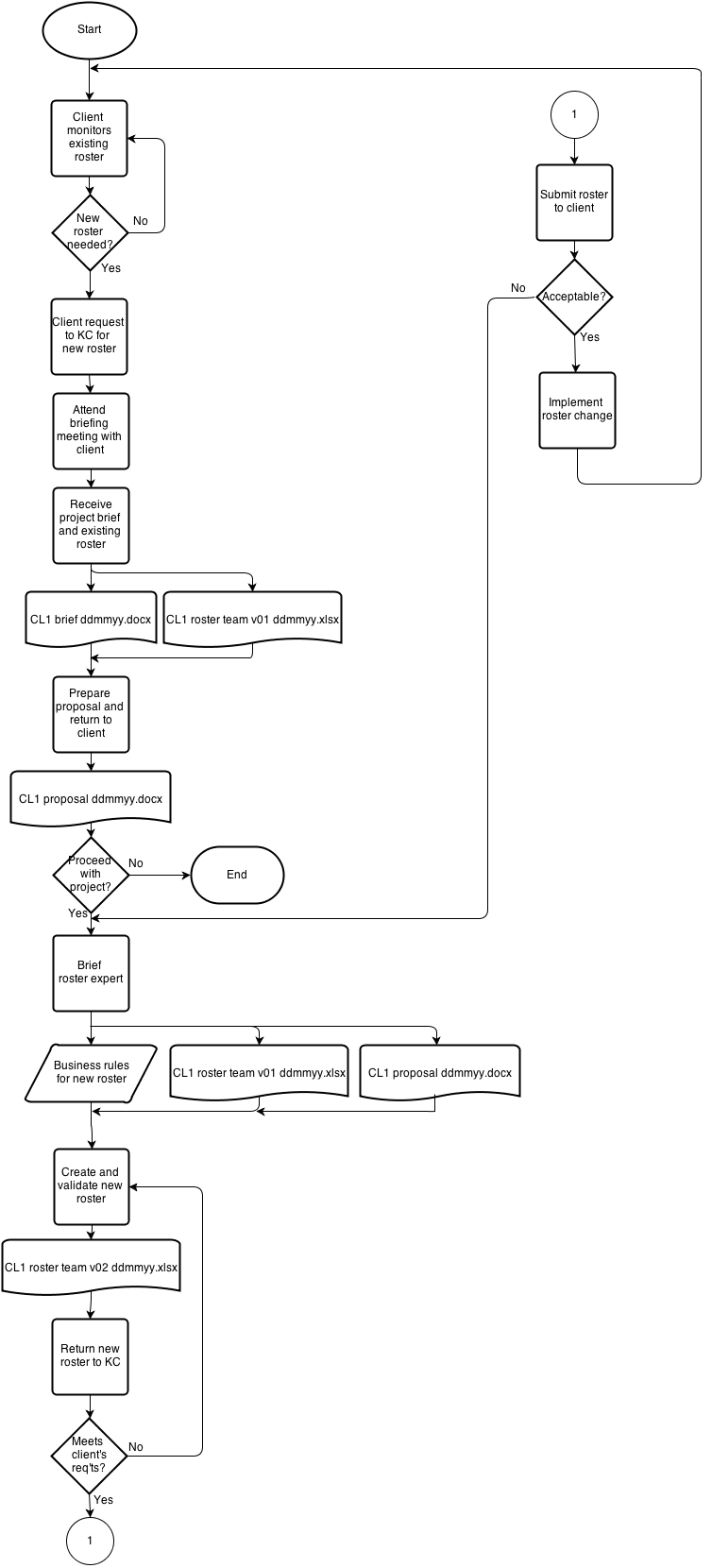
In order to solve this problem, a computer-based application would be produced to create the rosters themselves. The rules and limitations would be set by the consultant and then the program would do the bulk of the data processing to create and validate the rosters.

### Description of Existing System (inc Flow Chart)

From the interview and the other research I conducted, I found that the current process involves an operations manager at the client company, who requests a new roster proposal at least twice a year from Knatt Consulting. Knatt Consulting currently subcontracts the roster creation, hiring an expert who creates a proposal ‘by hand’ using spreadsheets. This is then validated (again by hand) at Knatt Consulting and submitted to the operations manager, who will introduce it and monitor its progress if it is acceptable.

It is up to the local duty managers to ensure that all shifts are covered even when staff are absent. The service managers have to review each request and check if there are enough staff to cover the employee who is on leave before modifying that week’s roster (for example, switching shifts). It is then the employees’ responsibility to work the modified shifts to suit the new arrangement. The roster is usually flexible enough to allow this.

The flowchart for the existing system is shown over the page.



### Systems outline chart (current system)

|  |  |
| --- | --- |
| **Input**  Request for new roster  Project brief & requirements | **Process**  Request for new roster  Brief  Validate roster  Return roster to client  Implement roster if successful |
| **Storage**  Proposal  Existing roster  New roster  Business rules | **Output**  New roster  Validated (final) roster |

### Users in the current system

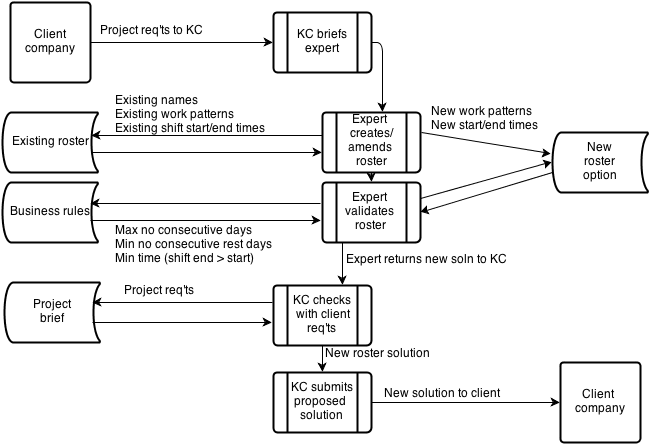
|  |  |  |
| --- | --- | --- |
| **Users** | | **Role** |
| **Organisation** | **Person** |
| Knatt Consulting | Consultant - Mr Knatt | Production, validation and submission of new roster proposals (twice yearly) |
| Rostering expert | Creates the roster proposal using data from Knatt Consulting |
| Client organisation | Operations Manager | Review potential roster changes and have Knatt Consulting modify roster to suit |
| Employees | Work the required shifts set up on the roster |
| Local Duty Managers | Ensure shifts are covered when staff are absent |

### Data Sources and Destinations

|  |  |  |  |
| --- | --- | --- | --- |
| **Current system** | **What is it?** | **Source** | **Destination** |
| Details of problem | Client company | Consultant |
| Project brief | Consultant | Roster expert |
| Existing shift details | Consultant/client company | Consultant/next roster proposal |
| New shift details | Consultant | Roster expert |
| Proposed solution | Consultant (with data from expert) | Client company |

|  |  |  |  |
| --- | --- | --- | --- |
| **Proposed system** | **What is it?** | **Source** | **Destination** |
| Details of problem | Client company | Consultant |
| Existing shift details | Client company/consultant | Consultant |
| Proposed solution | Consultant (with data from new roster) | Client company |

### Data Flow Diagram (existing system)

****

### Entity Relationship Diagrams

With a system that places heavy emphasis on databases, an entity relationship diagram would be needed to show the different users and data items and the relationships between them. Since this project does not seem to require significant data storage in this format, I have not produced an entity relationship diagram.

### Data volumes

The system I am to produce does not need to store large volumes of data. It will need to store created rosters and the details associated with them (including shift details, of which there will be a maximum of around 40 per day – 8 employees and 5 shift types). The roster creation part of the system only needs to run for one user at a time and therefore will not require significant data storage, so the overall volume of data stored will be low. However, the roster viewing extension has a higher volume of data involved since there will be the possibility for multiple user accounts, all of which will require differing shift patterns to be displayed. This is why the extension is designed as a web-based one: if implemented, this part of the system will have to deal with larger data volumes due to the increased number of users and the complexity of the data involved.

### Objectives for the Proposed System

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **System** | | **Processing** | | **User** | |
| S1 | The program must provide a complete solution to the roster creation (it must create and display rosters). | P1 | My system should process the data efficiently enough for it to create a roster in under two minutes when data is entered. | U1 | The user must be able to install the program from a memory stick in under five minutes on the target machine. |
| S2 | The system must allow a custom shift pattern to be entered in a form containing checkboxes to denote which shifts each employee can and cannot work. | P2 | My system’s data processing should be done so that it is error-free, producing rosters that exactly match the ones produced when the algorithm is traced by hand. | U2 | Once installed, the user must be able to access the program from the Start menu and the Desktop. |
| S3 | All inputs will have validation to ensure that only correct and useful data is entered. The user will be prompted when the checks fail. | P3 | The technique for processing of the rosters must include team days/public holidays. | U3 | Users should be asked to confirm whether they wish to exit before the program closes upon selecting ‘Exit’. |
| S4 | The system should allow a roster to be created for at least eight employees. | P4 | The system must process and export rosters as CSV files. | U4 | Users must be able to create a roster in under five minutes (including entry of parameters as detailed below) |
| S5 | The system should allow a roster to be created that is at least fifty days in length. | P5 | The roster files must be formatted into a non-software specific format that will allow them to be opened in a standard spreadsheet application. | U5 | The user must be able to enter penalties that are between 0 and at least 5000 for each rule. |
| S6 | The system must display rosters in a grid-style output. | P6 | The system must process and load created roster files into the grid in under twenty seconds on the target machine. | U6 | If the user enters a penalty of zero for a rule, that rule must not be applied to the roster creation. |
| S7 | The system should be able to email the created roster to each employee. |  | | U7 | Users must be able to open more than one roster at a time in the program to allow comparison. |
| S8 | The system should display the roster in different ways for different viewers (depending on if they are employees/managers). |  | |
| S9 | The completed installer and payloads for the program should take up less than 15MB of space on a memory stick. |
| S10 | The complete system will be less than 20MB, including the documentation and any example rosters I choose to include. |  | |
| S11 | The system must be able to be backed up onto a cloud-based storage system. |
| S12 | The system must make use of standard Windows ‘Open’/’Save’ dialogs for the file I/O parts of the program. |
| S13 | The system must feature key shortcuts (eg F1) to allow common tasks in the program to be accessed. |  | |
| S14 | The system must not conflict with existing software on target machines when in use. |
| S15 | The system should feature a colour scheme that ties in with the Knatt Consulting house style. |

#### Parameters for objective U4

The below parameters were later modified into a data set which was used for file testing. The results from this test procedure can be seen in the system testing section of this project and in appendix C.

|  |  |
| --- | --- |
| Number of days | 24 |
| Number of employees | 8 |
| Duplicate penalty | 5000 |
| Cover penalty | 5000 |
| Work days penalty | 500 |
| Rest days penalty | 300 |
| Fixed role penalty | 1000 |
| Following shift penalty | 50 |

### Potential Solutions

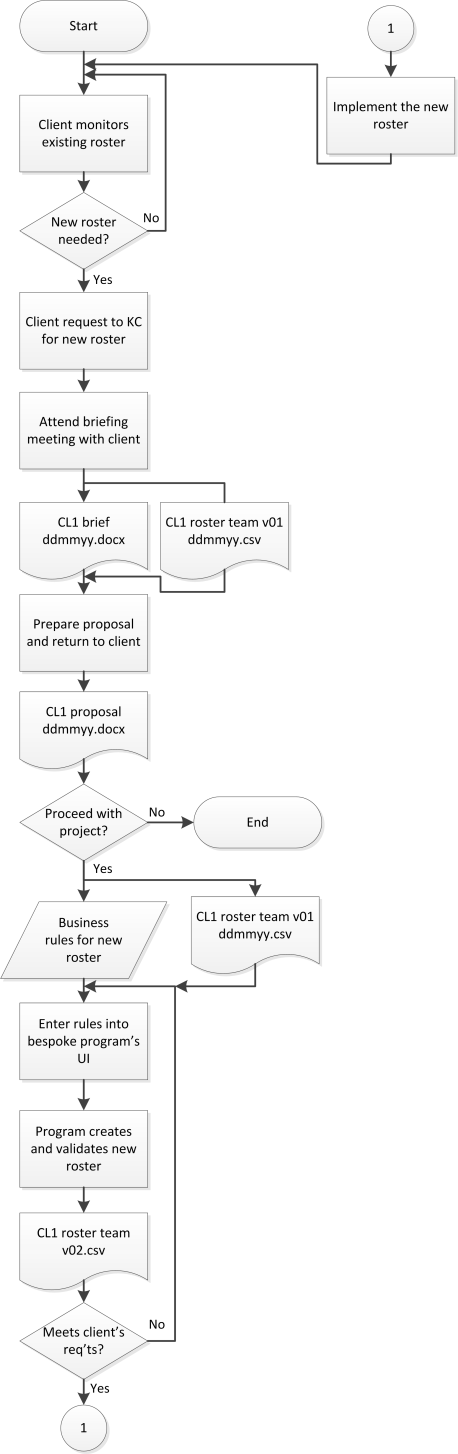
|  |  |  |
| --- | --- | --- |
| **Suggested solution** | **Positives** | **Negatives** |
| Creation of bespoke VB solution  One potential solution would involve the creation of a Visual Basic application to run on Mr Knatt’s PC. The Visual Basic application would create the rosters using an algorithm from information and rules which have been previously entered by Mr Knatt. The program would process the data input and use it to create a new roster; it would then be able to write the new roster to a file using a built-in function, for example.  If the project was extended to include the aforementioned proof of concept for a web-based system to display the timetables, Mr Knatt has a web server that could be used for this purpose (it currently is used to provide his website). | Visual Basic programming is something that I have reasonable experience with, making it easier to complete within the given time-frame.  A VB program would be easy to install, and would not conflict with the existing OS (Windows 7).  Creating a single program would tie in with Mr Knatt’s desire to ‘avoid purchasing bespoke software’, as discussed during the interview.  The sample of employees questioned responded well to the idea of a web-based extension of the roster system: it is likely that this sample represents the views of a significant number of the employees in the different client companies.  Mr Knatt already has a web server so if I did manage to complete the extension I would not have to find a new way of hosting it: I would be able to use his web space. | Creating an algorithm from scratch is very difficult, and using pre-made algorithms limits what I can do with the program.  In this case, it is unlikely that I will also have time to create the web-based front end, as programming a solution completely from scratch will take a significant amount of time.  The program would need to be extensively tested to ensure that it was reliable in all situations.  If the extension took place, producing a web solution that was capable of referencing a file written to in VB would be difficult; it would probably also require the creation of multiple user accounts for the different types of end users at the client company (as discussed in the interview). |
| Use of commercially available software solution  Another potential solution could involve the purchase or download of a rostering program (such as ABCRoster), and implementing it within the system. This would be installed onto Knatt Consulting’s PC and used to produce the rosters. Mr Knatt would still have to discuss his clients’ needs at meetings, as with the previous example. | This would reduce the time and effort needed to hand code a bespoke solution.  There would be little to no errors using this method as the program would have increased capabilities and has been tested to a greater level than what I can achieve in the given time frame.  This method still removes the subcontracting to the rostering expert and would enable Mr Knatt to produce the rosters much faster than his system currently does (since it involves computerisation).  This system could be implemented very quickly with very little extra coding needing to be done.  It is likely that the program would have significant documentation already, reducing the amount that I have to produce for the primary user. | This solution may not be problem-specific enough – hand coding a solution would allow me to design a program that exactly fulfilled the client’s needs. (I would not be able to modify existing, pre-compiled software and it may contain features that Mr Knatt does not want or need, or it could lack ones that he does need.)  Mr Knatt mentioned in the interview that he does not want to ‘purchase bespoke software’ and the abilities of free offerings may be questionable.  Additional research would have to be done into the different rostering programs that are available.  It would be very hard to modify this type of solution at a later date. |
| Creation of entirely web-based solution  An entirely web-based solution is another option that I have considered as part of my investigation. This would allow Mr Knatt to produce his rosters ‘online’ with the added benefit of the rosters that the application created being accessible from anywhere (as they would be ‘in the cloud’, and not stored locally as with option 1). Potential clients could be sent a direct link (via email) to the output from the rostering application.  Having this system entirely web-based would not require permanent installation. | If the hardware Knatt Consulting is using changes, the rostering application does not need reinstalling or changing as it is available on the company’s web server.  This system would be the easiest one to introduce the extended proof of concept of a web-based display into, as the file server is already being used to store and display the actual rostering application.  A web-based solution could be accessed from anywhere in the world using different devices.  The solution would run within the browser interface familiar to the end user, so Mr Knatt would not have to become accustomed to an entirely new environment. | A major problem with this approach is that it is very difficult to do significant data processing with a web-based application. It is also harder to install and maintain, and consequently could present problems if it needs to be modified at a later date.  People accessing this solution would all be using different browsers and devices and it is unlikely that I would be able to test it on all of the possible versions of these due to time constraints: it might therefore have a significant issue that is not found until after I have completed my part of the project.  This solution still requires the creation of an algorithm, meaning that it would take significantly more time to implement than option 2.  The web server may not be able to handle the increased resource load from the users. If it was to fail there would be no local backup of the created rosters. |

### Chosen Solution

After considering all of the solutions that I produced, I chose option 1 for further development. It seems to fit a higher proportion of the client’s requirements; I also have some skill with Visual Basic programming so it will be quicker to design and produce a basic prototype as I know how to use the development environment to a degree. The entirely web-based system would lead to problems with the data processing, which makes up a significant part of the system, and I am unsure how to handle the ‘saving’ of the rosters without writing files to the web server, which would put a higher load on the server’s resources. On the other hand, the introduction of bespoke software is attractive due to the lower potential of errors, and the fact that it would be easier to install and maintain. I have decided not to use this method as it will go against the idea of not ‘purchasing bespoke software’ and will not be specific enough for the client’s problem. However, all of the methods that I considered removed the need for subcontracting, which was one of the main objectives for this project.

The solution will require the local installation of a VB application which will replace the currently-subcontracted rostering expert. This will enable Mr Knatt to enter the rules that he receives from his clients directly into the VB application, instead of having to send them to the expert and wait for the production of rosters by hand. The program will produce and export a roster as a CSV, and it is likely that it will be able to validate rosters entered in a specific manner. Mr Knatt will then be able to send the proposed roster created by the program to his clients and get their feedback on the solution as before. As an extension, the web-based interface would be produced and hosted on Knatt Consulting’s web server and would read, interpret and display the CSV files, allowing them to be viewed by the client company’s employees. In conclusion, I feel that this solution allows for sufficient data processing capabilities, and will be easy to install and maintain due to the fact that it only needs to be provided on 1 machine. I have reasonable experience with VB and therefore would be able to update and modify the system after it is implemented (if necessary).

### Flow Diagram of Proposed System



### Evaluation of Analysis

In general I am happy with the results of my investigation into Knatt Consulting’s problem: the research methods I chose were useful in demonstrating to me that this was a viable problem for further investigation. I feel that the creation of the new system will be a great benefit to the company and will be more efficient and less error-prone than the current subcontraction. My research also provided me with an interesting possible extension – the improvement of the rostering program to have a web-based interface as a proof of concept. This could be implemented and incorporated if I have sufficient time. For now, the main focus of my project will be the design and implementation of the chosen solution (option 1 – the Visual Basic program running on Mr Knatt’s machine), bearing in mind the main objectives that the client has agreed to.

# Design

### Systems outline chart

This chart is a descriptor of the required inputs, outputs, processes and storage for the new system. This is done at a high level so that the whole system can be summarized. It is likely that these values could change as I develop the project.

|  |  |
| --- | --- |
| **Input**  List of employees’ details:   * Employee number (allocated by program) or name of employee (as an identifier) * Role * Current type of work * Specific issues regarding an employee   Limitations for new roster:   * Min. time between shifts * Shift length * Importance of different roles   Existing roster (if applicable) | **Process**  Create new roster from limitations/details  Validate roster  Export roster to file  View roster |
| **Storage**  To files:  Existing roster  New roster  Business rules | **Output**  New roster  Validated (final) roster |

### Full system top down design

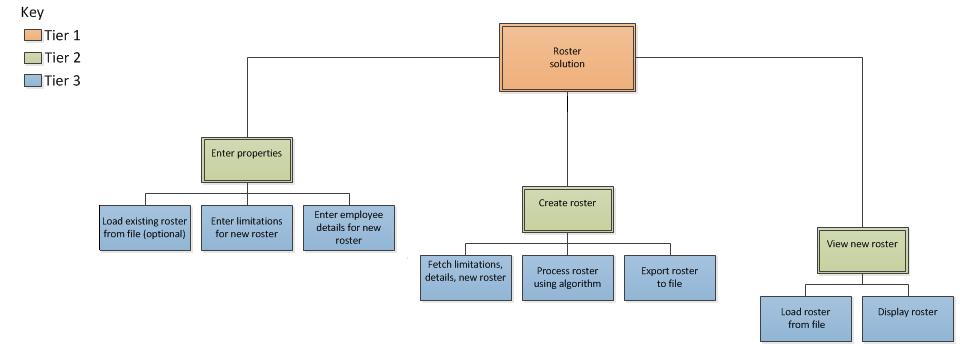
As I have to create a solution for a complex problem, my problem needs to be split into a modular form for ease of implementation. The top down design plan helps to display the different levels of complexity within my roster solution. The plan uses three levels of complexity (as shown by the key) and is a summary of the different tasks that the primary user will have to perform using the system.

The full top down design diagram is shown over the page due to the lack of space here.

### Entity-relationship diagrams

E:\COMP4\er diagram.png

The diagram above identifies the relationships between entities in the system, which would be used if I chose to solve the problem using a database. Multiple employees can be part of multiple rosters and one client company can have many rosters. Since there are many-one and many-many relationships, the data would require normalisation in a database-focused solution.



### Input design summary

Since the user is to input the majority of the data into the system, with a possible extension to cover an optional loaded roster from a previous scenario, the inputs to the system will be in string or integer form. All dates, time periods and importance of tasks will be entered as integer or decimal values. This could allow the algorithm to work on the roles with the greatest perceived importance first, or alternatively, depending on the algorithm I use, it may work through all employees and pick the best possible solution for each. Either way, the algorithm will make use of details such as the shift lengths and maximum time between shifts, along with other data (such as the total number of employees/total number of shifts, which are calculated after all the employees’ data has been entered).

### Output design summary

The system has one main output which is the completed roster. This will be saved to a file (CSV or text file) which will reside on the primary user’s hard drive or memory stick – it could be backed up to DropBox or a similar system to allow rosters to be accessed in the event of a hard drive failure, which is possible.

### Process design

The primary processing of data that needs to be done by the system will require an algorithm, as solving scheduling problems is extremely difficult without one. This project requires an algorithm that will assign each of the employees in the roster to specific shifts, and then order the shifts so that the most important tasks are covered first by these shifts, using the maximum time between shifts and the shift length to 'pack' the most important tasks first.

During the research and design of my project, I have considered multiple ways of solving the rostering and shift allocation problems that my project has presented me with. These include viewing the problem as a bin packing or knapsack problem (similar to those I have met on the Decision Mathematics course), using a ‘brute force’ method, or researching into other more complicated methods for data processing in rosters, such as the ‘GRASP’ approach.

#### Simple bin packing approaches

Two ‘bin packing’ approaches I have researched into in the design stage of this project are the ‘first-fit decreasing’ approach and the ‘full-bin packing’ approach. Both of these bin packing approaches view the roster as a set of ‘bins’: one bin would be the work requirement for each day. The employees are used to ‘fill’ the bins and would need ranking in order of importance for the tasks they work.

The first-fit decreasing bin packing algorithm takes a list of ranked items (with this particular problem, the employees and their tasks). This list can be in any order. The algorithm then orders the items in the list into descending order, with the highest ‘value’ or ‘rank’ task at the top. We then apply the ‘first-fit’ algorithm to this reordered list – that is, that each item is placed in the first available ‘bin’ that can take it. For example, if we had reached an employee and shift combination that was already present for the first ‘bin’ (day), then we would place this combination in the next ‘bin’ where the space would be available. The ‘bins’ and their contents would form the created roster. This algorithm has its advantages – it is quick, reducing processing time, and it is usually capable of producing an acceptable solution to the problem. Due to the sort method used being more efficient than the packing method, this algorithm is ‘dominated by the running time of first fit’ meaning that it will have a complexity of O(n2). However, the solution produced may not be ‘optimal’ (making use of the minimum number of ‘bins’), and so I also considered ‘full-bin packing’ in the research into bin packing for my project.

Full-bin packing uses observation to find combinations of items (the employee/task items as before) which will fill a ‘bin’. The items which can be combined to exactly *fill* a bin are packed into the ‘bins’ first. We then apply the descending first-fit algorithm to the remainder of the items in the list until all of the items are packed. This algorithm is much more likely to produce an optimal solution for the project, since the bins are more likely to be filled to maximum capacity (each day has the exact work requirement fulfilled). However, it is difficult to code and requires much more complicated logic than implementing the first-fit decreasing algorithm would. Below is a sample of pseudocode for first-fit decreasing bin packing, which would also be used as a part of a full-bin packing method.

First fit decreasing bin packing:

* Sort items with an appropriate sort method into decreasing order. Here the pseudocode for a quicksort is shown, but any appropriate sort method can be used.

*QUICKSORT(A,p,r)*

*If p < r*

*q = PARTITION(A,p,r)*

*QUICKSORT(A,p,q-1)*

*QUICKSORT(A,q+1,r)*

Partitioning routine within the quicksort has the following pseudocode:

*PARTITION(A,p,r)*

*x = A[r]*

*i = p - 1*

*for j = p to r-1*

*if A[j] <= x*

*i = i + 1*

*exchange A[i] with A[j]*

*exchange A[i+1] with A[r]*

*return i+1*

* Apply first-fit algorithm:

*For all objects i = 1,2, . . . , n do*

*For all bins j = 1,2, . . . do*

*If object i ﬁts in bin j then*

*Pack object i in bin j.*

*Break the loop and pack the next object.*

*End if*

*End for*

*If object i did not ﬁt in any available bin then*

*Create new bin and pack object i*

*End if*

*End for*

#### Knapsack problem approach

Considering my problem as a knapsack problem entails a much more complicated approach to solving it than the previous two ‘bin packing’ methods. However, through research I discovered this was possible using a dynamic programming approach.

The knapsack problem involves a list of items which each have a weight and a value. To solve the problem, one must work out how many of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is maximised. In the instance of my particular problem, the knapsack would be the roster, the weights of the shifts would be identical since there are only a set number of shifts to fill each day, and each shift would be given a value as before.

Since the shifts would be likely to have identical weights, but different values (or levels of importance), I decided that the knapsack approach would not be suitable for my project – the bin packing methods detailed previously would be much more suitable since the shift allocations that I must make do not fit the conventional ‘knapsack’ approach. Therefore, although this approach is likely to use a far more efficient algorithm than either bin packing method (due to the fact that dynamic programming would be likely to be used), I did not feel that doing any more research into solving knapsack problems was necessary.

#### Brute force approach

Using a brute force approach to solve the rostering problem is the simplest approach I considered, meaning that it is well within my capabilities as a programmer. However, using multiple levels of iteration, I felt that I would be able to adapt it to suit my problem, and I did more research into the most efficient ways of solving brute force problems.

Brute force algorithms are one of the slowest and most inefficient ways of solving allocation problems. A brute force algorithm considers every possible combination within a list of items, and calculates and stores a value for the suitability of each solution. It then selects the solution which is most suitable for the problem (usually the one with the lowest value). If I applied this method to my problem, the algorithm would need to run for every possible shift, for every staff member or employee, and on every single day of the roster. Having three embedded loops would mean it was more complex than the bin packing approach, and would give it a complexity equal to or greater than O(n3). This means that it would be inefficient in comparison and would take a lot longer to execute than the bin packing or knapsack approaches.

The brute force algorithm could be extended to use elements of AI by introducing extra statements into the algorithm. These might stop the algorithm from calculating the suitability for other combinations when an optimal solution has already been found, or the program could ‘learn’ which combinations produce the higher penalties and would skip these when it was used again in the future. This would allow a brute force solution to be produced that was more efficient, and the processing cost, although high at first, would drop as the program ‘learnt’ what was appropriate. It is worth noting, however, that these approaches would involve significantly more code and are still unlikely to be as fast as some other approaches I considered.

#### Hybrid ‘GRASP’ rostering approach

Upon reading a university paper by V. Cho, G. Wu and W. Ip of the Hong Kong Polytechnic University, I came across another method of ‘fully utilizing staff members to generate an effective roster’. The ‘GRASP’ rostering approach attempts to minimise the penalty generated by breaking ‘soft constraints’ – constraints that can be broken – whilst ensuring that ‘hard constraints’ are unbroken. The algorithm assigns staff shifts ‘greedily and randomly’ which means that different solutions can be found for the same data inputs. ‘Agents’ or staff members are then used to search all the solutions generated by the algorithm: when an agent finds a feasible solution, the other agents are informed, and all of the best solutions found so far are sent to one agent, who compares the quality of the rosters received and chooses the one with minimum penalty costs to be the final solution for the roster problem. I considered the last part of this approach to be similar to the previously-discussed brute force method, but the ‘GRASP’ approach uses a much more sophisticated primary algorithm to assign shifts before comparing the results. This approach could be applied to my project if I were to use one computer to compare the results at the end of the roster creation: the algorithm could be modified so that the computer was to choose the roster with the minimum penalty costs, rather than employing extra staff, since my roster problem is of a much smaller size than the one described in this paper.

Example pseudocode for the first part of the ‘GRASP’ method is detailed below. This is taken from the document detailed in the bibliography.

*Initialize all demandList according to the service demand*

*Initialize all tabuList*

*For each staff ‘Randomly pick a staff member*

*For each day ‘Randomly pick a day*

*pDayShift = retrieve yesterday’s shift*

*if demandList contains pDayShift then*

*if randomStart() then*

*tShift = pick a shift equal to pDayShift*

*tabuList.update()*

*end if*

*else*

*tShift = pick a shift obeying assignment rule*

*except pDayShift from demandList*

*tabuList.update()*

*end if*

*else*

*tShift = pick a shift obeying assignment rule*

*from shiftList*

*tabuArray.update()*

*end if*

*return tShift*

*next day*

*next staff*

### Identification of suitable algorithms for data transformation

From considering a series of algorithms during the process design stage, I have decided upon my algorithm for data processing within this system – the brute force method previously discussed. This is because it will provide thorough testing of every possible allocation and will be able to select the best possible solution based on the penalties it allocates. The brute force algorithm is within the scope of my programming skills, can be used to produce rosters that fit the client’s requirements, and is much easier to implement and test than one of the other algorithms that I have previously discussed. Therefore, I have decided that although the brute force method will be one of the less efficient ways of solving the rostering problem, it will be suitable for this project since it will be easier to produce a fully working solution that utilizes a brute force algorithm.

After identification of the most suitable algorithm for data transformation, I created a pseudocode ‘first draft’ of how the problem could be solved using a brute force method. This will later be extended into a fully working version in the development environment, and will also be used to produce trace tables by hand to check the accuracy of my computerized solution. This first draft is shown below.

### Example pseudocode of a suitable algorithm

This is an example of an early attempt at a brute force algorithm which runs for every day, staff member and possible allocation. It calculates the penalty for each possible allocation by working out which rules are violated. If a rule is violated, the penalty for that rule is added to the total penalty for this allocation. When all rules are considered, the total penalty for the shift allocation will be stored in an array or table. The possible allocations and their penalties will then be iterated through to find the first allocation with the lowest penalty (for example, if two allocations have the same penalty, the algorithm will select the first allocation with the lowest penalty). The algorithm then continues to find the next ‘optimal’ solution for the next staff member on the same day. When all staff members have had their shifts calculated, the algorithm continues to the next day in the roster. Once all days have had their shift pattern calculated, the algorithm is complete. The shift patterns for each day can then be output to the user, along with the penalties if required.

*For each day in roster*

*For each staffmember in day*

*For each possible allocation in staffmember*

*For each constraint in allocation*

*If rules violated then penalty points = current penalty + penalty(constraint number)*

*Next possible allocation*

*For each penalty in allocations*

*If penalty < current lowest penalty then current lowest penalty = penalty*

*Next penalty*

*For each penalty in allocations*

*Allocated shift = first shift with current lowest penalty*

*Next penalty*

*Next staffmember*

*Next day*

### Definition of data requirements (Design Data Dictionary)

A design data dictionary should include specific data types, and how they will be validated. These must be appropriate for how the data types are to be interpreted. It is entirely possible that these values will change during implementation, but this data dictionary provides a rough guideline as to the data items within the system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ref** | **Name** | **Data type** | **Length (char)** | **Occurences** | **Source of data** |
| 1 | Number of employees | Integer | 1-2 | 1 | User |
| 2 | Task type | Integer | 1 | 1 <= a <= emp. no | User |
| 3 | Number of tasks | Integer | 1-2 | 1-emp. no | User/program |
| 4 | Task importance | Integer | 1 | a | User |
| 5 | Shift length (hours) | Decimal | 1 | 1 | User |
| 6 | Min. time between shifts (hours) | Decimal | 1 | 1 | User |
| 7 | Employee ID | String | 1-128 char | 1-emp. no | User (if not assigned by program) |
| 8 | Employee comments | String | 0-128 char | 1-emp. no | User |

### Example validation required

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ref** | **Name** | **Validation checks** | **Regex** | **Description** | **Error message** |
| 1 | Number of employees | Presence, Length | Any number 0-9, length 2 digits  [0-9][0-9] | The number of employees in the roster. | Please enter a valid number of employees |
| 2 | Task type | Presence, Length | Any number 0-9, length 2 digits  [0-9][0-9] | The type of task that the employees must complete. | Please enter a number corresponding to task type |
| 3 | Number of tasks | Presence, Length | Any number 1-9, length 1 digits  [1-9] | The number of different types of task within the system. | Please enter the number of different roles (1-9) |
| 4 | Task importance | Presence, Length | Any number 1-9, length 1 digits  [1-9] | The importance of each different type of task. | Please enter the importance of each role (1-9) |
| 5 | Shift length (hours) | Presence, Length | Any number 0-9, length 2 digits, decimal  ^(0[1-9]|[1-9][0-9])$ | The length of each employee shift in hours. | Please enter a valid shift length |
| 6 | Min. time between shifts (hours) | Presence, Length | Any number 1-9, length 2 digits, decimal  [0-9]+(\.[0-9][0-9]?)? | The minimum time in hours that must be left between finishing one shift and starting the next. | Please enter a valid time between shifts |
| 7 | Employee ID | Presence, Length | Any number 1-9, length 2 digits  ^(0[1-9]|[1-9][0-9])$ | A unique identifier for each employee in the system. (Assigned an ID (number based) upon failing check) | Invalid employee ID entered, assigning numb er ID |
| 8 | Employee comments | None required | None required | Specific comments that the user has about each employee (aiding manual validation). | None required |

### Identification of storage media

Storing the program on a CD-ROM at first seems an attractive option as it provides a permanent copy of the program; however, the program will still require the capability to write files to its directory, and a CD-ROM would not allow this. Therefore, the principal storage media for my system will be the hard drive, or a memory stick.

Use of a USB memory stick will allow fast reading and writing of files, and installing the system onto the end user’s hard drive means that there will be a possibility for integration with other programs currently on the hard drive, and a higher security level (since it is more difficult to lose/damage a hard drive). Having the program on a memory stick will be convenient for Mr Knatt as it will be easily transferable between computers; however, it is worth considering that memory sticks are susceptible to loss or corruption.

Reasons for use of a memory stick for installation include:

* The file itself wouldn’t result in any wasted space on the flash drive, as the rest of the drive could still be read to and written from as usual.
* USB is extremely common on all laptops and desktops produced in the last ten years, and will continue to be in place on new hardware for a long time – this will ensure that the program can still be installed on newer hardware if necessary.
* Mr Knatt has several USB memory sticks which he already uses, and he could therefore use these for backup of the installation file if necessary.
* The cost to the company is reduced since the memory stick can be re-used for storage of other documents once the program is installed (unlike with a CD-ROM, for example).
* File transfer across USB 2.0 and 3.0 protocols is much faster than CD-ROMs.

The system will be installed on the primary user’s machine by myself, using a memory. The program, and its related files, will reside on Mr Knatt’s PC and will run from his machine, therefore the principal storage media for my system once it is installed will be the hard drive.

### Hardware specification

Since running an algorithm this complex in a VB environment will be processor-intensive, the machine needed to run the software will need to have a fast (preferably dual-core) processor.

A hard drive will be required both to store the program and the produced rosters, with the backup of both of these possible onto a USB memory stick. A keyboard and mouse is also required for input and interaction with the program. Screen display should be large (upwards of 800x600 pixels). Upon comparison with the primary user’s specifications, my program should run on the primary user’s hardware as it exceeds the minimum requirements.

|  |  |
| --- | --- |
| **Minimum requirement** | **Primary user’s hardware (from research)** |
| Intel Core 2 Duo or similar (>2GHz) | Intel Core i5 (2.40GHz) |
| 2048MB RAM | 4096MB RAM |
| 800x600 display resolution | 1333x768 display resolution |
| At least 128MB minimum storage (HDD/memory stick) | ~20GB available on HDD, access to several 4GB memory sticks for backup/transfer |

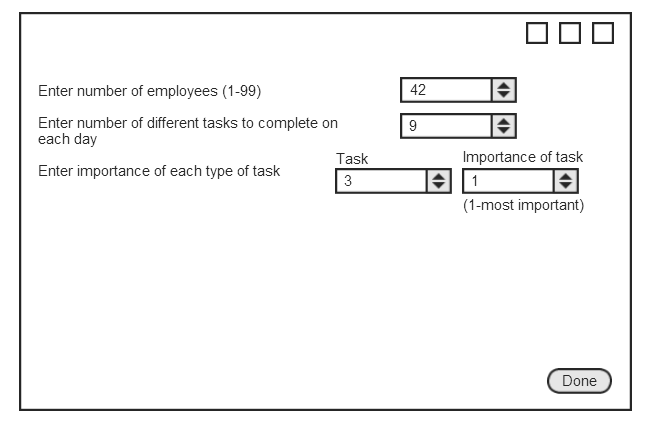
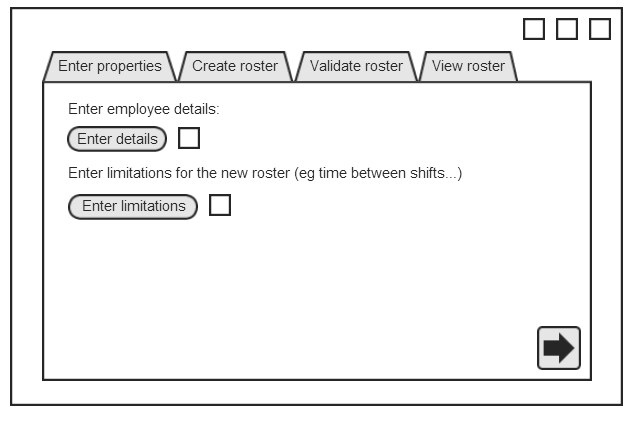
### User interface design (rationale)

The rostering system needs to have an easy-to-navigate UI, where the user performs the data entry and runs the different tasks from within the program in a logical sequence. This will be done by separating the different tasks that the program has to handle into ‘tabs’, or possibly a ‘wizard-style’ set of screenshots, and providing buttons on each to start the various processes. Since the data processing using the algorithm will be processor-intensive, and will take a significant amount of time, it is likely that I will implement a ‘loading’ status message to display to the user while this is taking place, or at least a message box to display to the user that ‘Roster creation is finished!’ or similar.

Sensible user interface design reduces the amount of errors that occur, and reduces the amount that the user has to remember. Formatting ‘hints’ will be necessary to reduce the risk that data entered fails the validation checks, as this will mean re-entry of that data. Choices will be made using controls such as drop-down lists, radio buttons or check boxes instead of using keyboard entry wherever possible, as this will also reduce the need for validation of the entered data. The error messages that my system produces also need to be relevant to the user’s level of understanding.

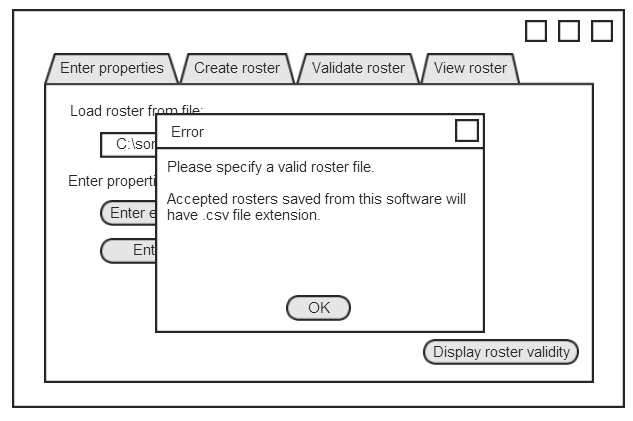
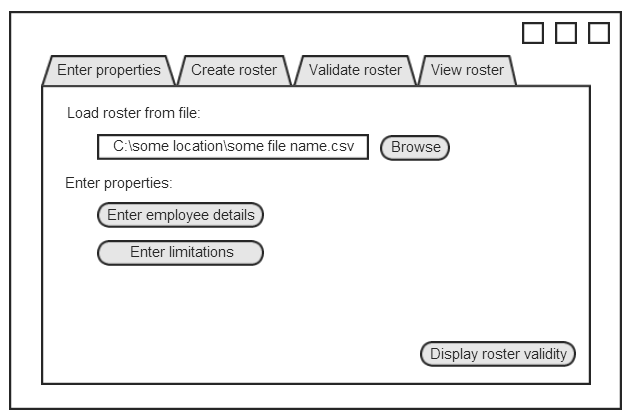
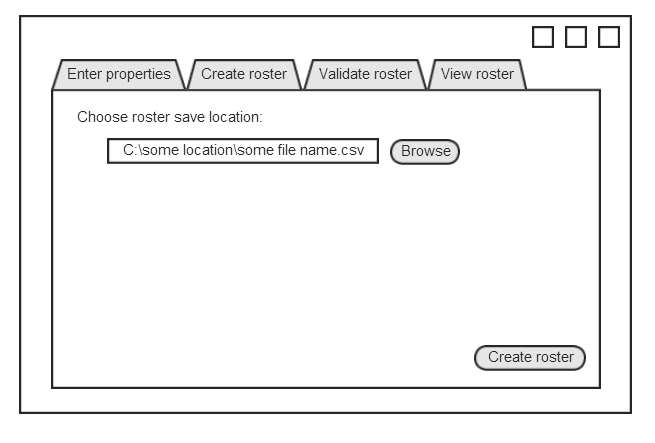
### UI sample of planned data capture and entry designs

Data capture/entry will be done through the Visual Studio text entry facilities, making use of regular expressions and validation to ensure that the user inputs valid data before it is processed by the algorithm.



Details entry – subset of properties entry. Regular expressions/validation methods used to ensure that valid data entry completed before dismissal is allowed. A help button (third from top right) is present on every page for the user to view the documentation if required.

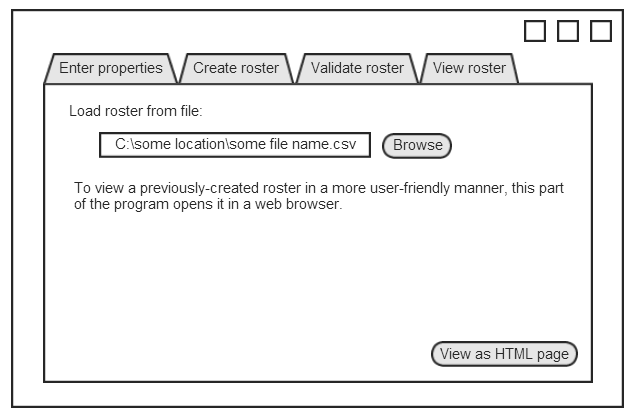
Properties entry. User enters employee details and limitations for the roster using the buttons. Icons next to buttons change to checkmarks when data entered correctly. User then chooses the arrow button to move to the roster creation part of the program.



Validation of existing roster. Example of error message that would result from invalid file detection (achieved through using regular expressions). This only permits the program to open the correct file type, as otherwise it will attempt to validate files which are not laid out in the same way, resulting in run-time errors.

Validation of existing roster. User enters file name and save location for an existing roster. The employee details/limitations are entered as before, then ‘Display roster validity’ checks the roster against the constraints entered, and displays a message box when processing is complete with the result of the checks.

Creation of roster. User enters file name and save location for the new roster. The ‘create roster’ button initiates the fetching of properties and the data processing within the program.



Roster viewing. As before, a previously-created roster is loaded from an existing save location before viewing it as an HTML page using Java or a similar method to interpret the CSV. This will produce a much more user-friendly table for the end user to read and interpret.

This could also be done within the program, if a suitable method is found.

### Sample of planned valid output designs

Outputs from the system will come in the form of status message boxes to update the user on the progress of roster creation, as well as the CSV or text file output from the roster creation part of the program. Below is a model of the CSV roster that the program is expected to produce.

The variables are output to the CSV for each task in the roster, with task1,2,3 being the unique identifier for each task, and days going from 1-length of roster. An optional note field could be included: this would allow notes to be added to the roster by the consultant regarding problems with each day, which the program will not identify. However, interpreting this in a ‘viewing’ interpreter would be difficult, and therefore it is likely that I will focus on the other parts of the program first.

#### Option one: ‘table’ inspired output

day,task1,task2,task3…taskn,note

1,day1task1,day1task2,day1task3,day1task(n),day1note

2,day2task1,day2task2,day2task3,day2task(n),day2note

3,day3task1,day3task2,day3task3,day3task(n),day3note

x,dayxtask1,dayxtask2,dayxtask3,dayxtask(n),dayxnote

Represented in a table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| day | task1 | task2 | task3 | task(n) | note |
| 1 | day1task1 | day1task2 | day1task3 | day1task(n) | day1note |
| 2 | day2task1 | day2task2 | day2task3 | day2task(n) | day2note |
| 3 | day3task1 | day3task2 | day3task3 | day3task(n) | day3note |
| x | dayxtask1 | dayxtask1 | dayxtask1 | day(x)task(n) | day(x)note |

#### Option two: simpler output

day1\_employee1/shift/note, day1\_employee2/shift/note, day1\_employee3/shift/note, …

This string would be repeated for each day, employee and shift assignment in the roster. Designing the file output in this manner is likely to make it easier for the program to read the data from the text file; however, it is difficult to understand, and the day value is written to file every single iteration of the output, when it does not need to be (compared to the first output design).

I plan to white box test both of these output file designs and select the more appropriate one for the system by comparing them against each other.

### System security and data integrity

The security of the system is important, as it could contain sensitive information regarding the employees or companies that the new system is being used to produce rosters for. To prevent unauthorised access, Mr Knatt’s PC already uses password protection, and he could consider installing some remote access software such as LoJack to lock his hard drive in the event that his computer is lost or stolen. To prevent other users using the program if it is placed on a memory stick, I shall implement some basic password protection for the program.

Security of data within the system is also important: since the program is self-contained, and the only file outputs are the completed rosters, it is unlikely that the user will have issues with this. Until the web extension is created, there will be a single user to the program, effectively an ‘admin’ – who will need to be able to enter, access and use all of the data in the system. If roster/shift display is created using a web-based interface it will be important to consider that employees may not be allowed to view other employees’ shifts and sensitive data, as this might incur issues with the Data Protection Act. To combat this, I would need to create user accounts with restricted privileges for the employees.

Data integrity also needs to be considered in the design of the system. Creating a system where there are many variables that need to be processed using an algorithm requires that the variables are in the correct format so they do not produce run-time errors when entered into the program. The previously described validation rules and regular expressions will attempt to make sure that the program has the correct data entered into all of its text fields.

If the program is corrupted, lost, or otherwise damaged so that it does not function properly, Mr Knatt will be able to reinstall it from the original data storage medium that I provide. It will be harder to recover created rosters in the event of a hard drive failure; however, it is likely that these will be backed up by the end user onto a cloud-based document server or a similar device in order to ensure that the rosters do not have to be created again.

### System testing – outline test plan

Below is the preliminary outline test plan for my created system. This uses different test methods to run through the system to test its basic functionality.

|  |  |  |
| --- | --- | --- |
| **Test** | **Description** | **Purpose of test** |
| 1 | Input testing | Validate inputs into the system using regular expressions/data validation. (bottom-up testing) |
| 2 | Flow of control testing | User’s path through the system is tested (should not be able to create a roster without entering data first) (top-down testing) |
| 3 | Algorithm and decision testing | The algorithm used in the system must produce the correct outputs into the roster. To test it a trace table should be completed (black box testing) |
| 4 | Testing saving of data | Data must be saved in the correct format/file type, with the correct file name (system testing) |
| 5 | Specification testing | Black box testing used to check that the system meets the original specification set out. |

### Alpha testing design (Input/Flow of control/File system testing)

This testing table provides detailed information on some of the alpha testing that needs to be done to the prototype of my system. Some variable names may change as I develop the solution to the project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Test Name | Purpose of Test | Data  Typical, Erroneous,  Extreme | Expected Outcome |
| 1 | Properties | To check valid properties entry. | TED1  EED1  XED1 | Accepted  Error message: ‘please enter valid properties as denoted in the dialog box’  Accepted |
| 2 | Limitations | To check valid limitations entered. | TLD1  ELD1  XLD1 | Accepted  Error message: ‘please enter valid limitations as denoted in the dialog box’  Accepted |
| 3 | Conflicts within system (1) | Checks if total number of users is higher than total job number. | No. users > No. jobs  No. users = no. jobs  No. users < No. jobs | Accepted  Accepted  Error message: ‘Please ensure that the number of employees is greater than the total number of jobs needed’  Accepted |
| 4 | Conflicts within system (2) | Checks for two shifts with the same weight. | TW1  EW1 | Accepted  Error message: ‘Please ensure that no two shifts have the same weight’ |
| 5 | Flow of control/  top-down testing (1) | On choosing ‘create roster’ both the properties and limitations must be filled in. | Properties and limitations complete  Properties incomplete  Limitations incomplete | Accepted  Error message: ‘Please enter all properties and limitations before creating the roster’ |
| 6 | Flow of control/  top-down testing (2) | On choosing ‘Validate existing roster’ at least one of the properties or limitations must be filled in. | Both complete  Both incomplete  One complete | Accepted  Error message: ‘Please enter constraints for the validation process’  Accepted |
| 7 | File system testing (1) | A correct file name must be entered that fits the regular expression for a file name before saving. | Save roster with name from TFD1  Save roster with name from EFD1 | Accepted, message: ‘roster saved to file’ or similar  Error message: ‘File name may not contain characters \\/:\\*\?""<>\|’ |
| 8 | File system testing (2) | The CSV display utility only opens files in CSV format. | Load or validate TFE1  Load or validate EFE1 | Loads correctly  Error message: ‘Please load a valid CSV file saved from this software’ |

These tables show the typical, erroneous and extreme values used to test my prototype. It is likely that the testing section shows different data sets as the outputs will have changed.

Data set: employee properties

|  |  |  |
| --- | --- | --- |
| Data set | Number of employees | Number of tasks |
| TED1 | 42 | 10 |
| XED1 | 90 | 90 |
| EED1 | 32478 | 101 |

Data set: limitations data

|  |  |  |
| --- | --- | --- |
| Data set | Min time between shifts | Shift length |
| TLD1 | 1 | 1.5 |
| XLD1 | 0 | 8 |
| ELD1 | 210 | 0.01 |

Data set: weights data

|  |  |  |
| --- | --- | --- |
| Data set | Shift number | Importance |
| TWD1 | 1  2 | 1  2 |
| EWD1 | 1  2 | 1  1 |

Data set: file data

Regex used: NOT [\\/:\\*\?""<>\|]

|  |  |  |
| --- | --- | --- |
| Data set | File name | Valid entry? |
| TFD1 | Roster1.csv | Yes |
| EFD1 | Roster\*\*?.csv | No |

Data set: file extensions

|  |  |
| --- | --- |
| Data set | File name |
| TFE1 | Roster1.csv |
| EFE1 | Roster1.xhtml |

### Alpha testing of algorithm and decision processes

Alpha testing of the algorithm used for roster production is done using a black box method where the inputs and outputs are recorded in a trace table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Sub-inputs** | **Input**  **Typical, Erroneous,**  **Extreme** | **Expected Output** |
| Properties | No. of employees | **42**  **53323**  **90** | **Roster with 42 lines**  **Does not pass validation**  **Roster with 90 lines** |
| No. of tasks each day | **12,**  **4574**  **99** | **Roster with 12 tasks filled each day**  **Does not pass validation**  **Roster with 99 tasks filled each day** |
| No. employees per task | **1-2, 2-82, 3-5**  **1-382, 2-324, 3-0**  **1-99** | **Roster with 2 employees on task 1, 82 on task 2 and 5 on task 3**  **Does not pass validation**  **Roster with 99 employees on task 1** |
| Importance of each type (weight) | **1-2, 2-1, 3-3**  **1-1, 2-1, 3-1**  **1-1, 2-2…99-99** | **Roster with task 1 as second priority, task 2 as first priority**  **Does not pass validation**  **Roster with task 1 as first priority…task 99 as 99th priority** |
| Limitations | Shift length (hours) | **1.5**  **0.002**  **4** | **A roster with 1.5 hour shifts**  **Does not pass validation**  **A roster with four hour shifts** |
| Min. time between shifts | **8**  **2343**  **1** | **A roster with a minimum of 8hrs between shifts Does not pass validation**  **A roster with a minimum of 1 hour between shifts** |

### Beta testing against specification

After alpha testing is completed, beta testing is used to check the prototype against the specification. This can be done by the end user and is a final chance for error-checking – the program will only have minor flaws by this stage if sufficient alpha testing has been completed.

The below table will be used in the testing section to determine whether or not the system I have produced meets the specification set out at the start of the project.

Evidence of this *completed* section is shown in the Appraisal part of the project.

|  |  |
| --- | --- |
| **System objective** | **Passed test? Y/N** |
| The program must provide a complete solution to the roster creation (it must create and display rosters). |  |
| The system must allow a custom shift pattern to be entered in a form containing checkboxes to denote which shifts each employee can and cannot work. |  |
| All inputs will have validation to ensure that only correct and useful data is entered. The user will be prompted when the checks fail. |  |
| The system should allow a roster to be created for at least eight employees. |  |
| The system should allow a roster to be created that is at least fifty days in length. |  |
| The system must display rosters in a grid-style output. |  |
| The system should be able to email the created roster to each employee. |  |
| The system should display the roster in different ways for different viewers (depending on if they are employees/managers). |  |
| The completed installer and payloads for the program should take up less than 15MB of space on a memory stick. |  |
| The complete system will be less than 20MB, including the documentation and any example rosters I choose to include. |  |
| The system must be able to be backed up onto a cloud-based storage system. |  |
| The system must make use of standard Windows ‘Open’/’Save’ dialogs for the file I/O parts of the program. |  |
| The system must feature key shortcuts (eg F1) to allow common tasks in the program to be accessed. |  |
| The system must not conflict with existing software on target machines when in use. |  |
| The system should feature a colour scheme that ties in with the Knatt Consulting house style. |  |

|  |  |
| --- | --- |
| **Processing objective** | **Passed test? Y/N** |
| My system should process the data efficiently enough for it to create a roster in under two minutes when data is entered. |  |
| My system’s data processing should be done so that it is error-free, producing rosters that exactly match the ones produced when the algorithm is traced by hand. |  |
| The technique for processing of the rosters must include team days/public holidays. |  |
| The system must process and export rosters as CSV files. |  |
| The roster files must be formatted into a non-software specific format, that will allow them to be opened in a standard spreadsheet application. |  |
| The system must process and load created roster files into the grid in under twenty seconds on the target machine. |  |

|  |  |
| --- | --- |
| **User objective** | **Passed test? Y/N** |
| The user must be able to install the program from a memory stick in under five minutes on the target machine. |  |
| Once installed, the user must be able to access the program from the Start menu and the Desktop. |  |
| Users should be asked to confirm whether they wish to exit before the program closes upon selecting ‘Exit’. |  |
| Users must be able to create a roster in under five minutes (including entry of parameters as detailed below) |  |
| The user must be able to enter penalties that are between 0 and at least 5000 for each rule. |  |
| If the user enters a penalty of zero for a rule, that rule must not be applied to the roster creation. |  |
| Users must be able to open more than one roster at a time in the program to allow comparison. |  |

# Testing

### Outline test plan

Below is reprinted the outline test plan from the design section of this project.

|  |  |  |
| --- | --- | --- |
| **Test** | **Description** | **Purpose of test** |
| 1 | Input testing | Validate inputs into the system using regular expressions/data validation. (bottom-up testing) |
| 2 | Flow of control testing | User’s path through the system is tested (should not be able to create a roster without entering data first) (top-down testing) |
| 3 | Algorithm and decision testing | The algorithm used in the system must produce the correct outputs into the roster. To test it a trace table should be completed (black box testing) |
| 4 | Testing saving of data | Data must be saved in the correct format/file type, with the correct file name (system testing) |
| 5 | Specification testing | Black box testing used to check that the system meets the original specification set out in the analysis. |

### Testing datasets

Whilst performing the five different tests listed in the outline test plan, I made use of several data sets for typical, erroneous and extreme data to simplify testing – the data sets contain information for all of the variables needed to produce a roster for the program.

#### Input testing datasets

Key:

|  |  |
| --- | --- |
| T\_D1 | Typical data set 1 |
| X\_D1 | eXtreme data set 1 |
| E\_D1 | Erroneous data set 1 |

Data set: employee/roster properties

|  |  |  |
| --- | --- | --- |
| **Data set** | **Number of days in roster** | **Number of employees** |
| TED1 | 24 | 8 |
| XED1 | 90 | 1 |
| EED1 | 324783434 | twelve |

Data set: penalties data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data set** | **Duplicate** | **Cover** | **Work days** | **Rest days** | **Fixed role** | **Following shift** |
| TPD1 | 5000 | 6000 | 500 | 300 | 1000 | 50 |
| XPD1 | 10000 | 10000 | 0 | 0 | 10000 | 0 |
| EPD1 | 123123 | 0.2 | 6238334 | -3 | 3.14159 | Three hundred |

Data set: min days worked/rested data

|  |  |  |
| --- | --- | --- |
| **Data set** | **Max. consec. work days** | **Min. consec. rest days** |
| TWD1 | 4 | 2 |
| XWD1 | 1 | 10 |
| EWD1 | -6.5 | Twelve |

### Input testing table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test purpose** | **Description** | **Data**  **Typical, Erroneous,**  **Extreme** | **Expected outcome** | **Actual outcome** | **Evidence in appendix** |
| 1.1 | Validate number of days. | Number of days should be accepted if it is between 1-90 inclusive. | TED1 – 24  XED1 – 90  EED1 - 324783434 | Accept 24  Accept 90  Error | Accepted  Accepted  Error | 1.1a  1.1b  1.1c |
| 1.2 | Validate number of employees. | Number of days should be accepted if it is between 1-8 inclusive. | TED1 – 8  XED1 – 1  EED1 – “twelve” | Accept 8  Accept 1  Error | Accepted  Accepted  Error | 1.2a  1.2b  1.2c |
| 2.1 | Validate duplicate penalty entry. | All rules should have an accepted penalty between 0-10000 inclusive. | TPD1 – 5000  XPD1 – 10000  EPD1 – 123123 | Accept 5000  Accept 10000  Error | Accepted  Accepted  Error | 2.1a  2.1b  2.1c |
| 2.2 | Validate cover penalty entry. | All rules should have an accepted penalty between 0-10000 inclusive. | TPD1 – 6000  XPD1 – 10000  EPD1 – 0.2 | Accept 6000  Accept 10000  Error | Accepted  Accepted  Accepted -required corrective action | 2.2a  2.2b  2.2c |
| 2.3 | Validate work days penalty entry. | All rules should have an accepted penalty between 0-10000 inclusive. | TPD1 – 500  XPD1 – 0  EPD1 – 6238334 | Accept 500  Accept 0  Error | Accepted  Accepted  Error | 2.3a  2.3b  2.3c |
| 2.4 | Validate rest days penalty entry. | All rules should have an accepted penalty between 0-10000 inclusive. | TPD1 – 300  XPD1 – 0  EPD1 – -3 | Accept 300  Accept 0  Error | Accepted  Accepted  Error | 2.4a  2.4b  2.4c |
| 2.5 | Validate fixed role penalty entry. | All rules should have an accepted penalty between 0-10000 inclusive. | TPD1 – 1000  XPD1 – 10000  EPD1 – 3.14159 | Accept 1000  Accept 10000  Error | Accepted  Accepted  Accepted – required corrective action | 2.5a  2.5b  2.5c |
| 2.6 | Validate following shift penalty entry. | All rules should have an accepted penalty between 0-10000 inclusive. | TPD1 – 50  XPD1 – 0  EPD1 – “Three hundred” | Accept 50  Accept 0  Error | Accepted  Accepted  Error | 2.6a  2.6b  2.6c |
| 3.1 | Validate max consecutive work days | Maximum consecutive work days must be between 1-10 inclusive. | TWD1 – 4  XWD1 – 1  EWD1 – -6.5 | Accept 4  Accept 1  Error | Accepted  Accepted  Error | 3.1a  3.1b  3.1c |
| 3.2 | Validate min consecutive rest days | Minimum consecutive rest days must be between 1-10 inclusive. | TWD1 – 2  XWD1 – 10  EWD1 – “Twelve” | Accept 4  Accept 1  Error | Accepted  Accepted  Error | 3.2a  3.2b  3.2c |

### Flow of control testing table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test purpose** | **Description** | **Test data** | **Expected outcome** | **Actual outcome** | **Evidence in appendix** |
| 4.1 | Check roster clearing. | A roster should not be able to be cleared if there is not one loaded. | We can clear a roster if there is one loaded  We can’t clear a roster if there isn’t one loaded | Roster can be cleared  Roster not able to be cleared (disabled) | Roster can be cleared  Roster not able to be cleared  (disabled) | 4.1a  4.1b |
| 5.1 | Check flow of control through forms. (1) | We should not be able to access createRoster2 unless createRoster1 is validated. | We can access createRoster2 with valid data on createRoster1 (TED1/XED1)  We can’t access createRoster2 if invalid data is entered  (EED1) | Able to access createRoster2  Not able to access createRoster2 (error message) | Able to access createRoster2  Not able to access createRoster2  (error message) | 5.1a  5.1b |
| 5.2 | Check flow of control through forms. (2) | We should not be able to access createRoster3 unless createRoster2 is validated. | We can access createRoster3 with valid data on createRoster2  (TPD1/XPD1)  We can’t access createRoster3 if invalid data is entered  (EPD1) | Able to access createRoster3  Not able to access createRoster3 (error message) | Able to access createRoster3  Not able to access createRoster3  (error message) | 5.2a  5.2b |
| 5.3 | Check flow of control through forms. (3) | We should not be able to create a roster unless createRoster3 is validated. | If valid data is entered on createRoster3, we are prompted for save location  (TWD1/XWD1)  If invalid data is entered, we are notified of the error  (EWD1) | Prompted for save location  Error message shown | Roster can be cleared  Roster not able to be cleared | 5.3a  5.3b |
| 5.4 | Check flow of control through forms. (4) | We should not be able to enter an invalid file name or save location. | Typical file name – ‘newRoster.txt’  Erroneous file name –“\*£$&(\*£&.txz” | Save successful  Save unsuccessful | Save successful  Save unsuccessful | 5.4a  5.4b |
| 6.1 | Menu testing - quit | Upon selecting ‘quit’ or pressing ALT-F4, the user should be prompted for confirmation. | N/A | Message box upon quit | Message box upon quit | 6.1 |
| 6.2 | Menu testing – create new roster | Upon selecting ‘create new roster’, createRoster1.vb should load | N/A | Form loads as expected | Form loads as expected | 6.2 |
| 6.3 | Menu testing – load roster (1) | Upon selecting ‘load roster from file’, we should be prompted for a save location | N/A | loadRosterDialog loads as expected | loadRosterDialog loads as expected | 6.3 |
| 6.4 | Menu testing – load roster (2) | If an illegal character is entered as a file name the program should notify the user. | Typical file name – ‘newRoster.txt’  Erroneous file name –“\*£$&(\*£&.txz” | Typical file name – ‘newRoster.txt’  Erroneous file name –“\*£$&(\*£&.txz” | Typical file name – ‘newRoster.txt’  Erroneous file name –“\*£$&(\*£&.txz” | 6.4 |
| 6.5 | Menu testing – clear roster (1) | Upon loading a roster, clear roster should be enabled and should prompt the user upon selection. | N/A | Enabled, prompt as expected | Enabled, prompt as expected | 6.5 |
| 6.6 | Menu testing – clear roster (2) | Roster should clear upon selecting ‘yes’ and then the option should be disabled since a roster is not able to be loaded. | N/A | Roster clears as expected, then option disabled | Roster clears as expected, then option disabled | 6.6 |
| 6.7 | Menu testing – ‘help’ (1) | Upon pressing F1 or selecting the menu option for ‘show documentation’ the document ‘example documentation for the roster solution.docx’ should be loaded. | N/A | Word document loaded | Word document loaded | 6.7 |
| 6.8 | Menu testing – ‘help’ (2) | Upon selecting the menu option for ‘Knatt Consulting on the web’ the correct web page should be loaded. | N/A | Web page loaded | Web page loaded | 6.8 |

### Iteration and decision testing

Iteration and decision testing has been done by hand using a large trace table which is attached as an appendix to the project. To do this I made use of white-box testing and created a new roster by hand, running the ‘algorithm’ in a spreadsheet using specific penalties for the rules and calculating the penalties for each allocation. I then recreated the scenario in the program I had created and ensured that they were the same.

See appendix C section 7 for a detailed trace table and the test roster created in the program.

### System testing

The system needs to be tested so that it writes files to the specified locations and the file contents are the same for different rosters created using the same parameters. This is a part of system testing.

To test the system, I used the ‘create new roster’ option within the program to create two rosters with identical parameters as shown below, one stored on a removable USB drive (E:\) and one in the Desktop directory of my computer. These were compared for identical contents.

The below parameters were modified from my original objectives which can be seen in the design section of this project.

*Roster parameters:*

|  |  |
| --- | --- |
| Number of days | 24 |
| Number of employees | 8 |
| Use custom shifts? | No |
| Duplicate penalty | 5000 |
| Cover penalty | 6000 |
| Work days penalty | 500 |
| Rest days penalty | 300 |
| Fixed role penalty | 1000 |
| Following shift penalty | 50 |
| Maximum consecutive work days | 4 |
| Minimum consecutive rest days | 2 |

### System testing table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Test purpose** | **Description** | **Test data** | **Expected outcome** | **Actual outcome** | **Evidence in appendix** |
| 8.1 | File check (1) | A roster saved at a specific file location should result in a file created at that location. | File save location: E:\txtRoster1.txt  File save location: C:\Documents and Settings\...\Desktop\txtRoster2.txt | Roster created at E:\txtRoster1.txt  Roster created at C:\Documents and Settings\...\Desktop\txtRoster2.txt | Both tests successful – files were created with the correct name at the correct location | 8.1a  8.1b |
| 8.2 | File check (2) | A roster saved with a specific file extension should have this extension when written to the storage media. | Roster file saved at E:\txtRoster1.txt  Roster file saved at C:\Docs and Settings\...\Desktop\txtRoster2.txt | Roster created at E:\txtRoster1.txt  Roster created at C:\Documents and Settings\...\Desktop\txtRoster2.txt | Both tests successful – files had the correct file extension (.txt) | 8.2a  8.2b |
| 8.3 | File contents check | Two rosters created using identical parameters should have identical contents. | Roster file saved at E:\txtRoster1.txt  Roster file saved at C:\Docs and Settings\...\Desktop\txtRoster2.txt | Both rosters have identical shift patterns and these are formatted in exactly the same way at the two different file locations. | Identical roster files produced with identical contents. | 8.3a  8.3b |
| 8.4 | File size check | The two rosters created should have identical sizes. | Roster file saved at E:\txtRoster1.txt  Roster file saved at C:\Docs and Settings\...\Desktop\txtRoster2.txt | Rosters are identical in size | Rosters are identical in size | 8.4a  8.4b |
| 8.5 | File extension check | The two rosters created should have identical file extensions. | Roster file saved at E:\txtRoster1.txt  Roster file saved at C:\Docs and Settings\...\Desktop\txtRoster2.txt | Rosters have identical file extensions | Rosters have identical file extensions | 8.5a  8.5b |

### System testing against specification

The final part of the testing procedure is done as black box testing against the user specification points, in order to confirm that the system meets the requirements set out at the start of the project. To do this, I showed the prototype to Mr Knatt for acceptance testing (client-specified).

Additional acceptance testing was performed as part of the Appraisal and can be seen in that section of the project.

|  |  |  |
| --- | --- | --- |
| **User objective** | | **Specification point passed?** |
| U1 | The user must be able to install the program from a memory stick in under five minutes on the target machine. |  |
| U2 | Once installed, the user must be able to access the program from the Start menu and the Desktop. |  |
| U3 | Users should be asked to confirm whether they wish to exit before the program closes upon selecting ‘Exit’. |  |
| U4 | Users must be able to create a roster in under five minutes (including entry of parameters as detailed in the design section) |  |
| U5 | The user must be able to enter penalties that are between 0 and at least 5000 for each rule. |  |
| U6 | If the user enters a penalty of zero for a rule, that rule must not be applied to the roster creation. |  |
| U7 | Users must be able to open more than one roster at a time in the program to allow comparison. |  |
| **End user signature:** | | C:\Users\James\Desktop\img009.jpg |

# Maintenance

### System overview

My system consists of a roster creation program and viewing utility. In roster creation, the user inputs the length of the roster, the number of employees, a custom shift pattern if appropriate, and other details concerning the penalty values for the ‘rules’ which the algorithm uses to form the ‘ideal’ solution for the situation. This is then exported as a text document and can be viewed within the program by the user.

The program contains two main sections: roster creation and roster viewing. Choosing the first option loads a wizard-style set of dialog boxes which are used to input the constraints for the roster. The user then selects a save location using a conventionally styled ‘save’ dialog box. Validation is in place throughout to ensure that values outside the range cannot be entered in each box, and if there is a problem with the save location the user will be notified (ensuring the system is robust in case it encounters any errors). The program is built on a complex ‘brute-force’ algorithm which tests every possible roster placement (according to the user entry) and uses this to create a solution for each employee on each day. It then calculates the penalty for each of these, and selects the first solution with the lowest penalty, writing this to the text file accordingly.

The second section of the program forms the ‘viewing utility’ which is also accessed from a drop down menu bar: here, the user enters the location on the file system from which to load a previously created roster text document, and the roster is loaded using a text file parsing subroutine within the program. This passes the data into a grid for the user to view a representation of the created roster. Loading different text documents gives the end user the capability to compare the benefits of different roster solutions created by the program.

The program effectively solves the problem that I set out during the investigation and analysis sections: it performs the tasks of roster creation and viewing with a good level of customization available to the end user. The program uses an algorithm which, although difficult to develop, will be reasonably easy to maintain: it will work with larger values as input than I have permitted, and therefore could be extended to cover a roster of more employees, or with more specific shift patterns, for example. It is quite processor-intensive in its current form, but the code I have produced enables future modification.

The program and associated resources come at a low cost to the end user: Mr Knatt already has a memory stick onto which I can install the files (if a new one needs to be purchased, these are inexpensive) and he is able to use Google Drive as a backup solution already, working extensively using the internet. The program could be considered to make use of heuristics: it is ‘able to produce a solution in a reasonable time frame that is good enough for solving the problem at hand’. The program does not guarantee to find the best possible solution, and currently it is up to the end user to select the roster with the overall lowest penalties after they have run the program multiple times. However, finding one acceptable solution to the problem is done in a sensible time frame, and although it will not be the ‘best’, it is likely (as in many of the scenarios I tested) to produce fully working rosters that could be introduced into a company workflow, just as the roster expert currently subcontracted to solve the problem does.

### Measures to ensure security of data

The security of the system that I have created has been considered: it contains specific information regarding the shift patterns and employee details of the companies that Mr Knatt is to create the rosters for. To ensure security of the system itself, the program is installed from a self-extracting zip file stored on a memory stick, to which only Mr Knatt has the password.

Security of data within the system is also important: to ensure this, the rosters created are saved into a password-protected folder which is synced to the chosen cloud-based file service – Google Docs: access to these backups is also password protected. This means that the client is protected in the event of a future failure, and the roster files themselves are protected from unauthorised users. Since the access to the rosters is password protected, I am happy that the created files will be sufficiently secure.

In the event of a hard drive failure, the program and rosters could be lost. If the program needs to be reinstalled, this can be done from the memory stick that it has been installed from. As previously mentioned, after discussion with the client, we have agreed to make use of Google Docs as a cloud-based file storage system – this enables the created rosters to be recovered.

### Summary of package(s) used

During the creation of this system, I chose to use Microsoft Visual Basic 2010 for the production of the code. This is because it is the programming language that I am most competent in: it enables good-quality user interface design using the Windows Forms elements, and also allowed me to use programming constructs such as Try…Catch statements, OOP (referencing items as new instances of objects) and elements such as the DataGridViewControl – an essential part of the solution I devised for roster viewing. As I was coding for a client who requires a Windows-based system, Visual Basic allowed me to produce an .exe file that, upon compilation, would be easy to install and would not require any reference to other programs to allow it to be run on the client’s machine, since the Visual Basic runtime libraries are included as a part of the Windows OS.

In addition to producing the main code for the system in Visual Basic, I have used Microsoft Word to create the ‘documentation’ file which can be loaded when the user chooses the ‘View documentation’ command in the program or presses F1, which is the default Windows help shortcut. This will open in the default Word document editor, which in this case is Word itself. If this system was transferred to another machine, it is likely that a user will have a program for editing word documents already installed (such as OpenOffice or LibreOffice). If not, I intend to include a simple ‘ReadMe’ text file that they can access for help.

During the creation of the part of the program that writes the roster to a text file, I made use of both Notepad and the free GPL licenced Notepad++ to aid me in producing a suitable file output. Using both of these text editors ensured that I could read the created roster file line by line in both and ensure that the file would be interpreted in the same way by each. Notepad++ is also capable of showing the line numbers of each line of a text document, and its total length. These were both features that I found very useful when designing the file output and working out how it would be read back into the program.

### Evidence of corrective, adaptive and perfective maintenance

Maintenance is necessary during the production and testing of the system to ensure that the solution that is produced is robust and as free from bugs as possible. I performed error testing on the solution, as demonstrated in the ‘System Testing’ section, to remove the flaws present in the system. I also sent the client several versions of the program whilst I was still in the process of creating it, to ensure that the solution I was working on would still meet the requirements set out by the client. Included in the appendix to this section are some emails exchanged with the client regarding this.

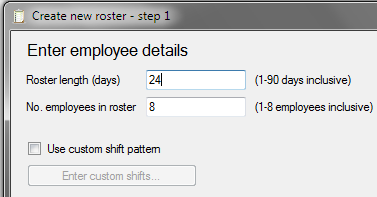
Different types of maintenance I have needed to perform on versions of the program are:

* Simple corrective maintenance: fixing incorrect labels or spelling errors, for example
* Adaptive maintenance: adding new code for validation, for example
* Perfective maintenance: adding new code to fulfil a user requirement

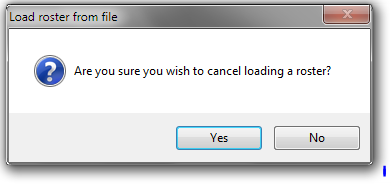
#### Simple corrective maintenance

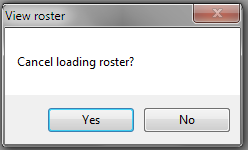
One example issue within the program that required simple corrective maintenance would be issues with some of the labels. The forms visible within the program often contained shortened labels that were used whilst the program was being coded. However, upon returning to the program at a later date, it was clear that the labels I had used did not aid the user-friendliness of the program. I also added default values into the text boxes on the forms which made testing whilst developing the program a lot easier.





(above: version 2 of the program, and version 15 of the program, showing the default values and new labels that I added in corrective maintenance)

Another issue that was corrected with simple corrective maintenance was the icons in some of the error message boxes. The inconsistency of the use of Windows forms icons (such as vbExclamation or vbInformation) in the message boxes could have been confusing to the user, and I felt that consistently making use of an icon in each message box would aid the user in understanding what was happening within the program.



(above: sample message boxes from version 4 of the roster solution and version 11, showing the icon use and improved message/title)

#### Adaptive maintenance

The text entry on the forms createRoster1.vb, createRoster2.vb and createRoster3.vb all initially used hard-coded ‘If’ statements for validation, along with Try…Catch constructs, as can be seen in earlier versions of the program. I later modified the data entry to use Regex validation alongside the Try…Catch statements, which will allow the program to be much more easily modified at a later date, as the programmer can simply change the regex string without having to modify an ‘If’ statement. Regex for string parsing is also likely to be faster to execute than an ‘If’ statement and, as such, I felt it was more appropriate for my solution.

This code is the validation function from createRoster1 as it was first implemented in version three.

It makes use of an ‘if’ statement (highlighted) to do the validation.

Function validateEntry(ByRef valid As Boolean)

valid = False

Dim errorMessageDisplayed As Boolean

'try...catch to catch any overflows or accidental text entry

...

'this If statement executes if the Try…Catch has passed, but is inefficient and hard to maintain

If (0 < days And days <= 90) And (0 < employees And employees <= 8) Then

valid = True

Else

valid = False

If errorMessageDisplayed = False Then

MsgBox("Please enter an integer day value between 1 and 90 inclusive, and an employee value between 1 and 8 inclusive.", \_

MsgBoxStyle.Exclamation, "Create new roster")

End If

End If

End Function

This code is the new validation function from createRoster1, used in later versions of the program. It makes use of the regex strings ‘dayRegex’ and ‘empRegex’ which produce matches (true/false) which are stored in ‘dayMatch’ and ‘empMatch’.

Function validateEntry(ByRef valid As Boolean)

valid = False

Dim errorMessageDisplayed As Boolean

Dim dayRegex As Regex = New Regex("^(90|[1-8][0-9]|[1-9])$") 'matches 1-90 inclusive

Dim empRegex As Regex = New Regex("^([1-8])$") 'matches 1-8 inclusive

Dim dayMatch, empMatch As Match

'try...catch to catch any overflows or accidental text entry

...

'regex match to check if the numbers are within set boundaries

dayMatch = dayRegex.Match(globalVariables.days)

empMatch = empRegex.Match(globalVariables.employees)

If empMatch.Success = True And dayMatch.success = True Then

'both of the inputs have passed the regex check

valid = True

Else

valid = False

If errorMessageDisplayed = False Then

MsgBox("Please enter an integer day value between 1 and 90 inclusive, and an employee value between 1 and 8 inclusive.", \_

MsgBoxStyle.Exclamation, "Create new roster")

End If

End If

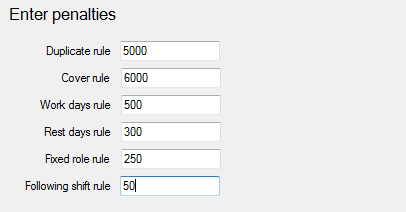
Return valid

End Function

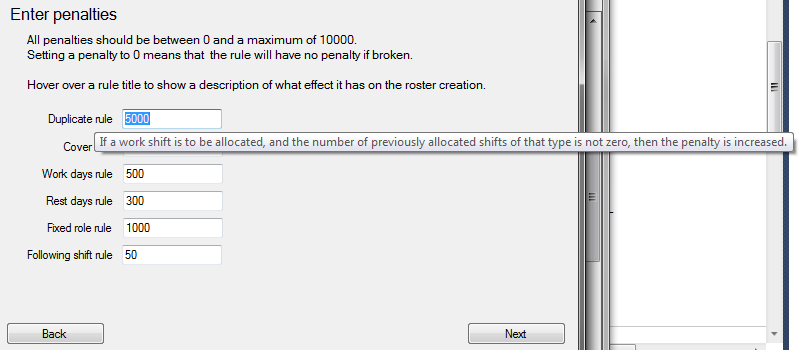
#### Perfective maintenance

An example of perfective maintenance within the solution came from a client request from email. After I had sent Mr Knatt version 11 of the program (in a .zip file by email), he responded with the following request: ‘could the rules that are hard coded be described somewhere on the screens so the user can see what the rule is before they enter the penalty?’. This led me to realize that I had not included any descriptions for any of the shifts or rules on either createCustomShifts1.vb or createRoster2.vb. To fulfil the user requirement, my perfective maintenance involved the introduction of ToolTips into the program to display messages to the user the different types of shift or penalty. I also edited the labels to inform the user of the presence of ‘descriptions’ of the shifts and penalties.

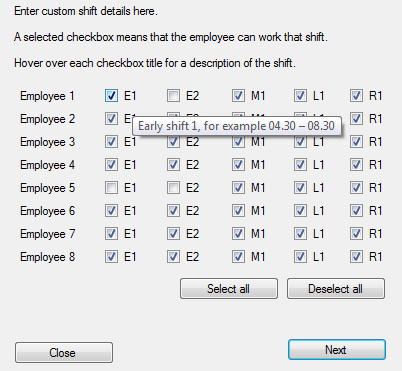
Original form, no labels:



Most recent version, showing descriptions, ToolTip explanations of the rules, and the text informing the user of how to show the ToolTips:

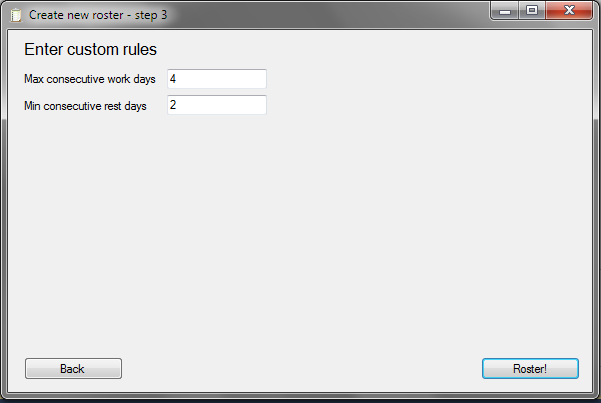


I also added ToolTips to the checkboxes on createCustomShifts1.vb:

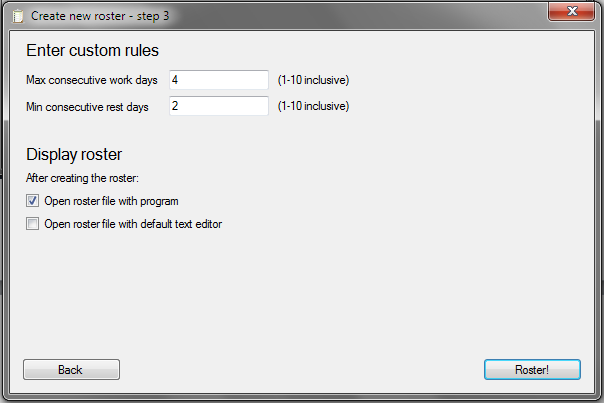


Mr Knatt also requested that I provided the option to open the roster file in the default text editor, in the program, or both – since previous versions of the program merely opened the text file that was created in Notepad for me to check whether the file output was correct, I felt that this change would be ‘slicker’ than before, since the end user is unlikely to want to open the roster in their text editor. By creating two new checkboxes and the associated code on createRoster3.vb, I was able to give the user the option to open the created roster in either, neither or both of these programs. I used the default ‘checked’ command for the ‘show roster in program’ checkbox to check the most commonly used option, enhancing the end user experience.

Early version of createRoster3.vb, showing the lack of checkboxes:



Most recent version showing the checkboxes, new descriptions and default values for the checkboxes.

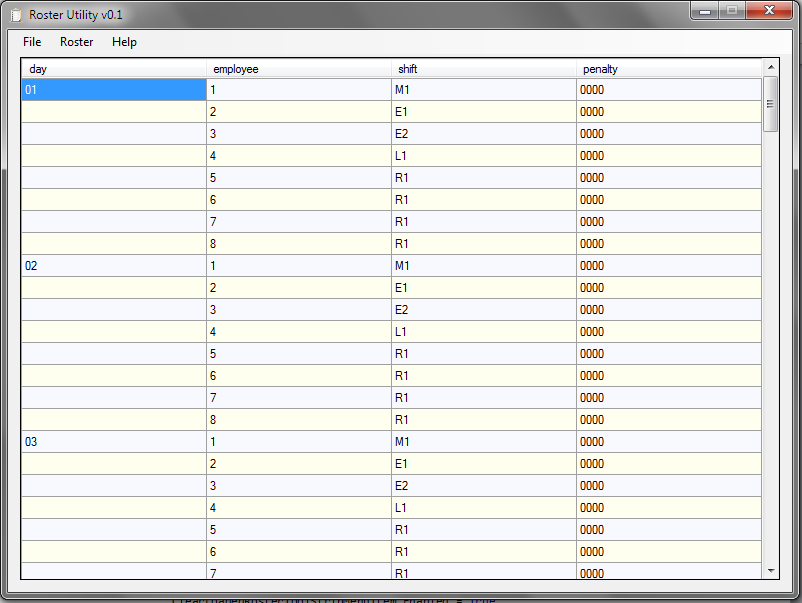


### Structural overview of the forms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Form** | **Role in system** | **Accessible from** | **Validation?** | **Processing?** |
| mainForm.vb | This is the first form that the user sees when they run the program. From here the different parts of the program are accessed using a menu control. This form also contains the rosterGridView where the roster is loaded into. | Always visible in the background – accessible from any form if the currently-opened form is closed. | Yes:  The loading and saving of files is validated using a try…catch construct. | Yes – upon loading a roster the text file is processed as strings and loaded into rosterGridView. |
| createRoster1.vb | This is the first stage of the ‘wizard style’ roster creation part of the program. Here, the user enters the number of employees in the roster, and the roster length, and can select to use a custom shift pattern. Clicking next runs the validation check and, if successful, stores the user entered data in the globalVariables module. | mainForm.vb  createRoster2.vb (upon selecting ‘back’ button control) | Yes, using regex.  Regex:  ^(90|[1-8][0-9]|[1-9])$ for roster length (1 to 90 days)  ^([1-8])$ for employee number  (1-8 employees) | No |
| useCustomShifts1.vb | This is a small dialog box which is accessed via a button on createRoster1.vb. Here the user enters the required custom shift pattern using the checkbox controls and submits them to be saved using a button. | createRoster1.vb (upon selecting ‘Use custom shifts’ button control) | No, since checkboxes do not require typed data entry/validation. | No |
| createRoster2.vb | This is the second stage of roster creation. Here, the user enters the penalties for each rule in the roster. Clicking next runs the validation check and, if successful, stores the user entered data in the globalVariables module. | createRoster1.vb (upon selecting ‘Next’ button control)  createRoster3.vb (upon selecting ‘Back’ button control) | Yes, using regex.  Regex:  ^(10000|[0-9]{0,4})$  (values from 0-10000 inclusive) | No |
| createRoster3.vb | This is the final stage of roster creation. Here, the user enters the maximum/minimum number of days worked/rested for each employee. They also select whether or not to open the created roster in their text editor and/or the program itself . Clicking next runs the validation check and, if successful, stores the user entered data in the globalVariables module. | createRoster2.vb (upon selecting ‘Next’ button control) | Yes, using regex.  Regex:  ^(10|[1-9])$  (used to validate both of the text fields, numbers 1-10 inclusive)  The program also checks for file errors when saving or trying to open rosters using Try…Catch constructs. | Yes: when the user selects the relevant button control and selects a save location, the roster is created using the algorithm. |

### Sample screenshots of forms

Below is a screenshot of mainForm.vb with a roster loaded into the rosterGridView control.



Penalties column of rosterGridView, showing data read from text file

Scrollbar allows user to scroll to the bottom of rosterGridView

Different colours used on alternating lines of the gridView to make the roster easier to read

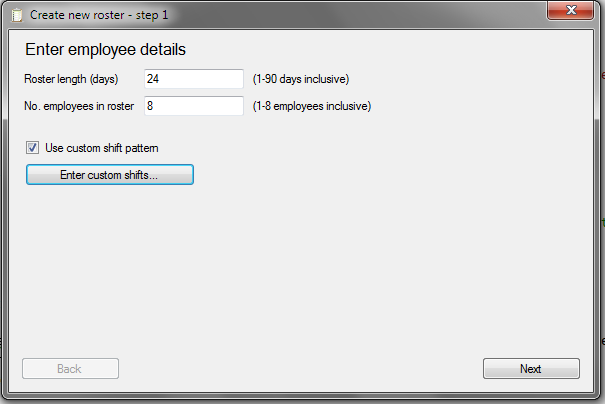
File menu, containing ‘Exit’ command

Roster menu, used to load, create or clear a roster

Help menu, used to load documentation or view Knatt Consulting website

Two loops used so this value is only displayed for the first employee on each day

This is createRoster1.vb – the first form loaded when the user selects ‘create a new roster’ or presses the F5 key.



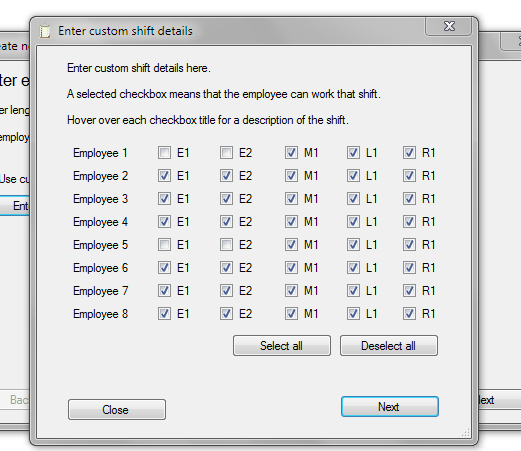
Checking this checkbox enables the ‘Enter custom shifts’ button which loads the form where a custom shift pattern will be entered.

User enters roster length and number of employees which is validated using a try…catch statement and regex.

This loads createCustomShifts1.vb

‘Next’ runs the validation statements and will permit the user to move on if their data entry matched the regex.

This is createCustomShifts1.vb – where the custom shift pattern is entered. It is accessed from the ‘enter custom shifts’ button on createRoster1.vb.



Description so that the user is aware of how to enter a typical shift pattern.

The form loads with a default shift pattern each time.

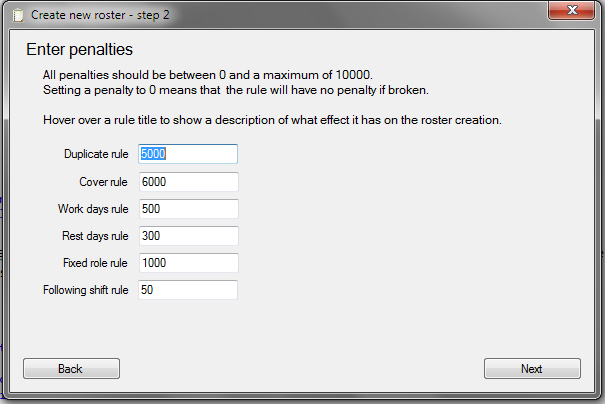
Select/Deselect all – once chosen, the buttons are disabled (since you can’t deselect all without anything selected).

Choosing ‘Close’ confirms that the user wants to stop entering custom shifts, then closes the form and sets the value for ‘use custom shifts’ to false.

‘Next’ saves the values for the checkboxes to an array of shifts, informs the user that the shifts are saved, and closes this form.

Each checkbox title has a tooltip displayed upon hover which shows a description of that shift.

This is createRoster2.vb, where the penalties are entered for the different rules. Hovering over a rule title will show the user its description.



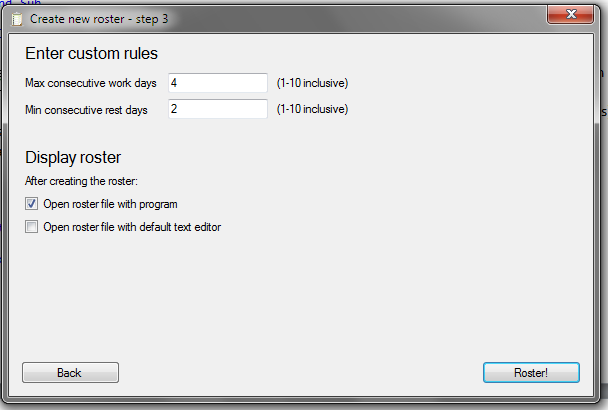
Penalties are entered in these input boxes, and are validated by regex, similar to createRoster1.vb

Closes this form and loads createRoster1.vb

Will allow the user to move on if all penalties pass validation. They are saved into an array.

Description of penalty entry.

This is createRoster3.vb, where the user enters the maximum/minimum consecutive work/rest days, and how they want to display the roster. These are validated using Regex and selecting ‘Roster!’ will begin the algorithm, which is detailed later in this section.



Closes this form and loads createRoster2.vb

User enters max/min number of days worked/rested which is validated using a try…catch statement and regex.

These checkboxes allow the user to decide whether to open the roster file in the rosterGridView on mainForm.vb, or in their default text editor.

Runs validation checks and, if successful, prompts the user for a save location, and creates the roster using the algorithm. The program then opens the roster in the required format and closes this form.

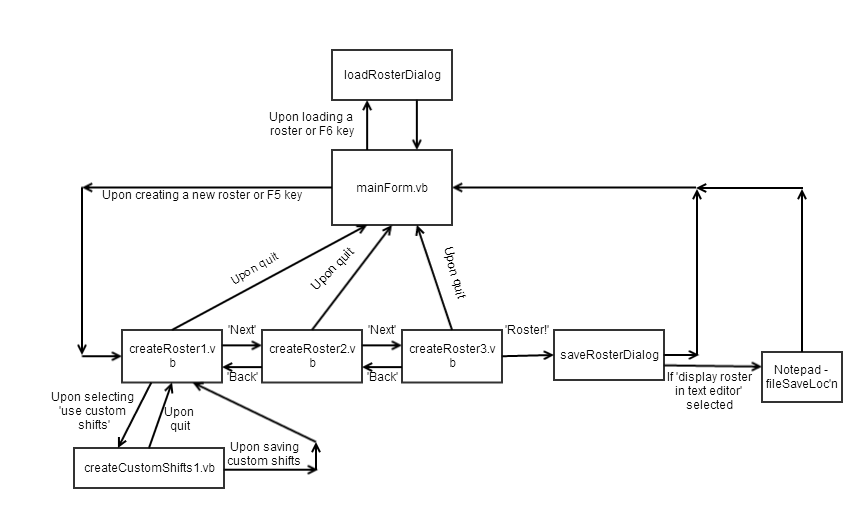
### Structure diagram of the forms – top down design

Runs validation checks and, if successful, prompts the user for a save location, and creates the roster using the algorithm. The program then opens the roster in the required format and closes this form.

These checkboxes allow the user to decide whether to open the roster file in the rosterGridView on mainForm.vb, or in their default text editor.

Closes this form and loads createRoster2.vb

Below is a simple structure diagram showing how the forms in my system fit together, and the various paths through them.



### List of all the variables used in the system: name, access, type and description

#### createCustomShifts1.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| toExit() | exitResponse | Public | Integer | Value: True/vbYes if user cancels shift entry |
| nextButtonClick() | matches() | Private | Array of Control | Contains a list of all checkboxes to match |
| employees | Private | Integer | Count variable – 1-8 – loops through all checkboxes on the form |
| j | Private | Integer | Count variable – 1-2 – used to switch between E1 and E2 by concatenation |
| cb | Private | CheckBox | Checkbox container used in a DirectCast command to find the position and number of checked checkboxes |
| clearCustomShifts() | i | Public | Integer | Count variable – 0-7 – used to replace each element in customShifts |
| j | Public | Integer | Count variable – 0-4 – used to replace each element in customShifts |

#### createRoster1.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| Global declarations in createRoster1.vb | valid | Public within class | Boolean | Program allows the user to move onto the next form if this variable is true |
| validateEntry() | errorMessageDisplayed | Public | Boolean | Set to true if a MsgBox with the appropriate error message has previously been displayed |
| dayRegex | Public | Regex | Holds regex ^(90|[1-8][0-9]|[1-9])$ which is used to validate a day value from 0-90 |
| empRegex | Public | Regex | Holds regex ^([1-8])$ which is used to validate an employee number from 1-8 inclusive |
| dayMatch | Public | Match | Success if day regex is passed |
| empMatch | Public | Match | Success if employee regex passed |
| customShiftsEntryButton\_Click | createCustomShiftsForm | Private | createCustomShifts1 | Used to create a new instance of createCustomShifts1 with createRoster1 set as the owner |

#### createRoster2.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| Global declarations in createRoster2.vb | valid | Public within class | Boolean | Program allows the user to move onto the next form if this variable is true |
| validateEntry() | errorMessageDisplayed | Public | Boolean | Set to true if a MsgBox with the appropriate error message has previously been displayed |
| penaltyRegex | Public | Regex | Holds regex ^(10000|[0-9]{0,4})$ which is used to validate a penalty value from 0 to 10000. |
| penaltyMatch | Public | Match | Success if penalty regex is passed |

#### createRoster3.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| Global in module | noDays | Public | Integer | Number of days entered on the form of createRoster1.vb |
| noEmployees | Public | Integer | Number of employees entered on createRoster1.vb |
| duplicatePenalty | Public | Integer | Penalty for failing the duplicate rule (entered on createRoster2.vb) |
| coverPenalty | Public | Integer | Penalty for failing the cover rule (entered on createRoster2.vb) |
| workDaysPenalty | Public | Integer | Penalty for failing the work days rule (entered on createRoster2.vb) |
| restDaysPenalty | Public | Integer | Penalty for failing the rest days rule (entered on createRoster2.vb) |
| fixedRolePenalty | Public | Integer | Penalty for failing the duplicate rule (entered on createRoster2.vb) |
| followingShiftPenalty | Public | Integer | Penalty for failing the following shift rule (entered on createRoster2.vb) |
| fileSaveLocation | Public | String | Contains location for saving file, entered into saveRosterDialog |
| maxConsecutiveWork | Public | Integer | Holds maximum consecutive days to work, entered on createRoster3.vb |
| minConsecutiveRest | Public | Integer | Holds minimum consecutive days to rest, entered on createRoster3.vb |
| valid | Public | Boolean | Set to true to allow the roster to be created |
| quit | Public | Boolean | Holds user response if the program is to be quit. (true if they want to quit) |
| possibleAllocations() | Public | Array of String() | 1D array that holds strings containing shift names (E1, E2…) |
| noShiftsPrevAllocated(,) | Public | Array of Object(,) | 2D array holding a string and integer to represent the shift and the associated number of times it has been allocated |
| prevDayAllocation(7) | Public | Array of String() | 1D array holding the previous day’s allocation for each employee |
| noConsecutiveDaysWorked() | Public | Array of Integer() | 1D array holding the number of consecutive days worked for each employee |
| noConsecutiveDaysRested() | Public | Array of Integer() | 1D array holding the number of consecutive days rested for each employee |
| totalNoShiftsWorked(7) | Public | Array of Integer | 1D array holding the number of shifts worked for each employee |
| totalNoShiftsRested(7) | Public | Array of Integer | 1D array holding the number of shifts rested for each employee |
| shiftPosition | Public | Integer | Holds the position of the previously allocated shift |
| maxNoEmployees | Public | Integer | noEmployees – 1 |
| penalty | Public | Integer | Used to hold the penalty for the shift that is currently being calculated |
| shiftAllocated | Public | String | Holds the current shift allocated, eg “E1” |
| minPenalty | Public | Integer | Holds the minimum penalty for the current shift allocation |
| allocationMinPenalty(,) | Public | Array of Object(,) | Contains the shiftAllocated value (string) and the minPenalty value(integer) |
| allocationsPenalties(,) | Public | Array of Object(,) | Contains each possible shift allocation (string) and its penalty (integer) |
| validateEntry | errorMessageDisplayed | Public | Boolean | Used in validation – set to true if an error message has already been displayed |
| workRestRegex | Private | Regex | Holds the regex ^(10|[1-9])$. True if the string is a number between 1-10 inclusive. |
| workMatch | Private | Match | True if the value for maxConsecutiveWork passes the check in workRestRegex |
| restMatch | Private | Match | True if the value for minConsecutiveWork passes the check in workRestRegex |
| findAllocationMinPenalty | finalcount | Public | Integer | Used to find first allocation with minimum penalty. |
| Count | Public | Integer | Used to find first allocation with minimum penalty. |
| writeAllocationsToFile | daysToWrite | Public | String | Holds the day number to write to text file, for example 01, 18 |
| penaltyToWrite | Public | String | Holds the penalty to write to text file, for example 0000, 0500 |
| shiftToWrite | Public | String | Holds the shift to write to text file, for example E1, M1 |
| saveRoster | response | Public | Integer | Used to hold the value of the user response in saveRosterDialog (save or cancel) |
| cancelRoster | Public | Integer | Used to hold the value of the user response from the prompt to confirm the roster creation is to be cancelled |

#### globalVariables.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| Global within module | days | Public | Integer | Number of days in the roster |
| employees | Public | Integer | Number of employees in the roster |
| penalties(5) | Public | Array of Integer() | Contains the values that the user entered as penalties on createRoster2.vb |
| customShifts(7,4) | Public | Array of Boolean() | This is an array of True/False values to determine which checkboxes the user has ticked and therefore which shifts are permitted to be worked |
| useCustomShifts | Public | Boolean | True if custom shifts are to be used |
| loadSilently | Public | Boolean | True if a roster is to be loaded without using the openRosterDialog control (ie the value for fileSaveLocation is already known) |

mainForm.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| Global within module | splitAtSlashes() | Public | Array of String() | Holds each element of the text file after it has been split using splitRoster |
| fileSaveLocation | Public | String | Holds the location of the saved roster |
| rosterLoaded | Public | Boolean | True if roster is loaded |
| toExit() | exitResponse | Public | Integer | Holds user response to ‘confirm quit’ dialog |
| dataGridRosterView\_MouseUp | cms | Public | ContextMenuStrip | Holds the context menu items to display when the dataGridView is right clicked |
| item1 | Public | cms.Item | Menu item for ‘clear loaded roster’ |
| item2 | Public | cms.Item | Menu item for ‘load new roster’ |
| menuChoice | item | Private | Ctype(sender,toolStripMenuItem) | New type declaration for the items in the ContextMenuStrip labelled ‘cms’ |
| selection | Private | CInt(item, Tag) | Type declaration for the selection that the user makes from ‘cms’ |
| loadRoster | response | Public | Integer | Holds the response from loadRosterDialog (open or cancel) |
| cancelRoster | Public | Integer | Holds the response from the prompt confirming whether to quit program |
| readTxtRoster | firstLineOfRoster | Public | String | Holds the first string read from the file – the actual roster data |
| numberOfEmployeesInRoster | Public | String | Holds the second string read from the file – the number of employees |
| r | Public | StreamReader | Declaration for the StreamReader used to read the text file at fileSaveLocation |
| fillDataGridView | list | Public | List (of rosterGridView) | List container declaration to contain the rows to add to the dataGridView named ‘rosterGridView’ |
| j | Public | Integer | Used to repeatedly loop through each employee in the roster, resets at end |
| clearRoster | clearResponse | Public | Integer | Confirms that the user wishes to clear the roster (true or false) |

#### rosterGridView.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sub/Function** | **Variable name** | **Access** | **Type** | **Description** |
| Global in module | day | Public | Property of rosterGridView | Column descriptors used for the dataGridViewControl named ‘rosterGridView’.  These are created using the Get/Set commands on rosterGridView.vb |
| employee |
| shift |
| penalty |

### List of all the procedures/functions used in the system

#### mainForm.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Procedure/**  **Function** | **Access** | **Description** | **Parameters** | **Outputs** |
| mainForm\_Load | Private | This sub runs when mainForm loads and is used to check if a roster is loaded, and changes the drop down toolbar item by calling changeDropDown. | sender, e | N/A |
| changeDropDown | Public | This sub is used to change the drop-down toolbar item when the roster is loaded. | rosterLoaded | N/A |
| toExit | Public | This sub confirms if the user wishes to exit the program and will close the form if necessary. | N/A | N/A |
| dataGridRosterView\_MouseUp | Public | This sub handles a user clicking on the rosterGridView and will show a context menu with the appropriate items if the mouse click is a right-click. | sender, e | N/A |
| menuChoice | Private | This sub handles the events that run when a user selects the different items in the context menu created by dataGridRosterView\_MouseUp. | sender, e | N/A |
| loadRoster | Public | This sub shows the ‘open file’ dialog box which is contained in a control called loadRosterDIalog. It uses try…catch constructs and validation for presence checks before a valid file name is passed back into the program. | fileSaveLocation, loadSilently | fileSaveLocation |
| readTxtRoster | Public | This sub uses a new streamReader object to read from the roster stored at fileSaveLocation. It then runs the subs splitRoster, fillDataGridView, and changeDropDown. | fileSaveLocation, rosterLoaded | rosterLoaded |
| splitRoster | Public | This sub splits the string containing the first line of the roster between each slash delimiter and stores the resulting elements in the array ‘splitAtSlashes’ | firstLineOfRoster, splitAtSlashes | splitAtSlashes |
| fillDataGridView | Public | This sub loops through each employee and day in the roster and writes each element of splitAtSlashes into the rosterGridView control. Two loops are used so that the day value is only written once for each day in the roster, and not for each employee. | splitAtSlashes, numberOfEmployeesInRoster | N/A |
| clearRoster | Public | This sub confirms that the user wants to clear the roster. If so, the dataGridRosterView dataSource is set to “nothing” and the drop-down menu is changed using changeDropDown. | rosterLoaded | N/A |
| ExToolStripMenuItem\_Click | Private | This sub handles the user choosing the ‘exit’ command from the toolStrip control. It will call toExit if this happens. | sender, e | N/A |
| CreateNewRosterToolStripMenuItem\_Click | Private | This sub handles the user choosing the ‘Create new roster’ command from the toolStrip control. This sub shows the first form for roster creation – createRoster1.vb | sender, e | N/A |
| loadExistingRosterToolStripMenuItem\_Click | Private | This sub handles the user choosing the ‘Load existing roster’ command from the toolStrip control. It calls the loadRoster sub which will load a new roster. | sender, e | N/A |
| clearLoadedRosterToolStripMenuItem\_Click | Private | This sub handles the user choosing the ‘clear loaded roster’ command from the toolStrip control. It calls clearRoster which will clear a roster if one is loaded. This item is enabled/disabled by the program - a user can only clear a roster if one is loaded. | sender, e | N/A |
| ViewDocumentationToolStripMenuItem\_Click | Private | This sub handles the user choosing the ‘view documentation’ command from the toolStrip control. It uses a try…catch construct to open a word document containing example documentation for the program. | sender, e | N/A |
| KnattConsultingOnTheWebToolStripMenuItem\_Click | Private | This sub handles the user choosing the ‘Knatt Consulting on the web’ command from the toolStrip control. This sub uses a try…catch construct to open the Knatt Consulting website in the default web browser. | sender, e | N/A |

#### createRoster1.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Procedure/**  **Function** | **Access** | **Description** | **Parameters** | **Outputs** |
| nextButton1\_Click | Public | This sub checks if the user has entered valid data using validateEntry and then calls the nextForm sub to change forms. | sender, e | N/A |
| validateEntry | Public | This function uses regex validation to check if the data entered into the text boxes on the form is valid. A try…catch construct prevents errors such as text entry or numerical overflow. The function returns true or false depending on if the data is valid. | valid | valid |
| nextForm | Public | This sub closes createRoster1 and loads createRoster2 | N/A | N/A |
| customShiftsEntryButton\_click | Private | This sub handles the user clicking on the ‘use custom shifts’ button and creates a new instance of createCustomShifts1.vb with createRoster1 as the owner. | sender, e | N/A |
| useCustomShiftsCheckBox\_CheckedChanged | Private | This sub handles the user checking the custom shifts checkbox. It enables/disables the customShiftsEntryButton and sets the value of globalVariables.useCustomShifts to true or false appropriately. | sender, e | N/A |

createCustomShifts1.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Procedure/**  **Function** | **Access** | **Description** | **Parameters** | **Outputs** |
| selectAllButton\_Click | Private | This sub loops through every checkbox on the form and sets their values to checked. It also disables the select all button and enables the deselect all button. | sender, e | N/A |
| deselectAllButton\_Click | Private | This sub loops through every checkbox on the form and sets their values to unchecked. It also enables the select all button and disables the deselect all button. | sender, e | N/A |
| closeButton\_Click | Private | This sub handles a user closing the custom shifts form and calls the toExit sub. | sender, e | N/A |
| toExit | Public | This function confirms that the user wishes to stop entering custom shifts and sets the useCustomShifts value to false. | N/A | N/A |
| nextButton\_Click | Private | This sub handles a user clicking the next button. It loops through all the checkboxes on the form and writes their value to the globalVariables.customShifts array. It then sets useCustomShifts to true. | sender, e | N/A |
| clearCustomShifts | Public | This sub loops through each element in customShifts and sets its value to false. | customShifts | customShifts |

#### createRoster2.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Procedure/**  **Function** | **Access** | **Description** | **Parameters** | **Outputs** |
| nextButton1\_Click | Public | This sub checks if the user has entered valid data using validateEntry and then calls the nextForm sub to change forms. | sender, e | N/A |
| backButton1\_Click | Public | This sub calls the prevForm sub to change forms. | sender, e | N/A |
| validateEntry | Public | This function uses regex validation to check if the data entered into the text boxes on the form is valid. The entered data is read into the penalties array and each element is checked against a regex string. A try…catch construct prevents errors such as text entry or numerical overflow. The function returns true or false depending on if the data is valid. | valid | valid |
| nextForm | Public | This sub closes createRoster2 and loads createRoster3. | N/A | N/A |
| prevForm | Public | This sub closes createRoster2 and loads createRoster1. | N/A | N/A |

#### createRoster3.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Procedure/**  **Function** | **Access** | **Description** | **Parameters** | **Outputs** |
| validateEntry | Public | This function uses regex validation to check if the data entered into the text boxes on the form is valid. The entered data is checked against a regex string. A try…catch construct prevents errors such as text entry or numerical overflow. The function returns true or false depending on if the data is valid. | valid | valid |
| createRosterButton\_Click | Public | This sub handles the user clicking on the ‘Roster!’ button. It checks if the user has entered valid data using validateEntry and then calls the processRoster sub. | sender, e | N/A |
| processRoster | Public | This sub loads all required variables from globalVariables.vb. It then calls saveRoster to prompt the user for a save location. If the user response is not ‘quit’, this sub then calls createNewRoster, and runs the appropriate sub to load the roster into the program or into a text editor depending on which checkboxes are checked. It then runs clearAll to clear all variables,and closes createRoster3. | N/A | N/A |
| createNewRoster | Public | This sub uses the algorithm detailed in the later section to loop through each possible shift combination for every employee on every day, calculate the penalties for each (using calculatePenalty), and pick the first allocation with the lowest penalty (using findAllocationLowestPenalty). It then increments the number of consecutive days worked/rested as appropriate, and calls writeAllocationsToFile to write the allocation to the text file.  After looping through all employees and all days, the sub then writes the end of roster, number of employees, and total number of days worked/rested by each employee to the text file. | N/A | N/A |
| calculatePenalty | Public | This function runs for each allocation for each employee for each day and calculates the penalty for each allocation. It returns the penalty value. | penalty, employees, count | N/A |
| findAllocationLowestPenalty | Public | This function loops through each allocation for each employee and finds the first allocation with the minimum penalty. It returns this allocation. | allocationsPenalties(,), minPenalty, allocationMinPenalty(,) | minPenalty, allocationMinPenalty |
| writeAllocationsToFile | Public | This sub writes the allocations for each employee to the text file at fileSaveLocation using a StreamWriter. It ensures that the day and penalty values are always the same length by adding the correct number of leading zeros based on their length. | days, employees, allocationMinPenalty, fileSaveLocation | N/A |
| writeDaysWorkedToFile | Public | This sub is run at the end of the roster creation and is used to write the number of days worked/rested by each employee to the text file using a StreamWriter. | totalNoShiftsRested, totalNoShiftsWorked, fileSaveLocation, employees | N/A |
| clearAll | Public | This sub resets each variable to 0 or “” and clears the arrays by looping through each of their elements. | N/A | N/A |
| saveRoster | Public | This function uses the saveRosterDialog object on the form to create a validated save dialog box. The file will be overwritten if required, and the file location is written to fileaveLocation. The function returns the save location. | fileSaveLocation, quit | fileSaveLocation, quit |
| openRoster | Public | This sub uses a Try..Catch construct to open the roster txt file in the default text editor. | fileSaveLocation | fileSaveLocation |
| backButton2\_Click | Private | This sub handles the user clicking ‘back’. It closes the current form, and loads createRoster2. | sender, e | N/A |

#### globalVariables.vb

This module file does not contain any procedures or functions.

#### rosterGridView.vb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Procedure/**  **Function** | **Access** | **Description** | **Parameters** | **Outputs** |
| New | Public | This sub creates a new object of class rosterGridView with the properties detailed within the form. It also alternates the datagrid colour styles to make a loaded roster easier to read, using cellstyle properties. | day, employee, shift, penalty | N/A |

### Algorithm overview

This project makes use of an O(n3) algorithm: there are three embedded loops since the brute force ‘penalty calculation’ runs for every shift allocation, for every employee, and for every day. The algorithm looks to see how many shifts worked/rested by each employee there already have been, and the problems with assigning certain shifts on certain days, and assigns a penalty according to the rules and constraints. This is done for every possible allocation for one employee at a time. After the penalty for all possible allocations for each employee are calculated, the first allocation with the lowest penalty is found (for example, if two allocations have a penalty of 0, the first shift will always be selected). This is then allocated and the possible allocations for the next employee on the same day then have their penalties calculated. Over the page are some flowcharts of how the penalties are calculated dependent on the rules and on the shifts previously worked, both on the current day and previous days in the roster.

### Pseudocode of the created algorithm (modified from Design section)

*For each day in roster*

*For each employee in day*

*For each possible allocation*

*For each rule in allocation*

*If rules violated then penalty points = current penalty + penalty(constraint number)*

*Next possible allocation*

*For each penalty in allocations*

*If penalty < current lowest penalty then current lowest penalty = penalty*

*Next penalty*

*For each penalty in allocations*

*Allocated shift = first shift with current lowest penalty*

*Next penalty*

*Open file for writing*

*Write employee, allocation, shift*

*Clear all variables*

*Next staffmember*

*Next day*

*Write number of employees*

*For each employee in roster*

*Write number of days worked/rested*

*Next employee*

### Coded functions/subroutines for the created algorithm (from above pseudocode)

This is the code that was created as an adaptation of the above pseudocode. It is run from the sub processRoster() which is called upon the user clicking ‘Roster!’ on createRoster3.vb after the entered data has successfully passed validation tests. The full integration of these subs/functions into the program can be seen in Appendix B: they are commented to make the adaptation from the pseudocode clearer.

Public Sub createNewRoster()

'for each day in the roster

For days = 1 To noDays

'for each employee on the day

For employees = 0 To noEmployees - 1

'calculate penalty for each possible allocation

For count = 0 To 4

allocationsPenalties(count, 1) = calculatePenalty(penalty, employees, count)

Next

'find first allocation with lowest penalty

findAllocationLowestPenalty(allocationsPenalties, minPenalty, allocationMinPenalty)

'set the allocated shift to be the allocation with minimum penalty

shiftAllocated = allocationMinPenalty(0, 0)

'if a work shift is allocated, the number of work days is increased and the number of rest days reset

'if a rest shift is allocated, the number of rest days is increased and the number of work days reset

If shiftAllocated <> "R" Then

noConsecutiveDaysWorked(employees) = noConsecutiveDaysWorked(employees) + 1

noConsecutiveDaysRested(employees) = 0

Else

noConsecutiveDaysRested(employees) = noConsecutiveDaysRested(employees) + 1

noConsecutiveDaysWorked(employees) = 0

End If

'save the new allocation (shiftAllocated) into shiftPrevAllocated (shift allocated to preceding employees

'ON THE CURRENT DAY) so that it is not assigned again unless needed

shiftPosition = Array.IndexOf(possibleAllocations, shiftAllocated) 'finds position of the previously allocated shift

noShiftsPrevAllocated(shiftPosition, 1) = noShiftsPrevAllocated(shiftPosition, 1) + 1

'save shiftAllocated into prevDayAllocation(employee)

prevDayAllocation(employees) = shiftAllocated

'write the shift allocation for this employee to file

writeAllocationsToFile(days, employees, allocationMinPenalty, fileSaveLocation)

Next 'next employee

'reset the variables which are only used on one day

shiftAllocated = ""

noShiftsPrevAllocated = {{"E1", 0}, {"E2", 0}, {"M", 0}, {"L", 0}, {"R", 0}}

For employees = 0 To noEmployees - 1

If prevDayAllocation(employees) = "R" Then

totalNoShiftsRested(employees) = totalNoShiftsRested(employees) + 1

Else

totalNoShiftsWorked(employees) = totalNoShiftsWorked(employees) + 1

End If

Next

Next 'next day

'write end of roster to file

Using writer As New IO.StreamWriter(fileSaveLocation, True)

writer.WriteLine("&&&")

End Using

'write no of employees to file

Using writer As New IO.StreamWriter(fileSaveLocation, True)

writer.WriteLine(noEmployees)

End Using

'write total number of days worked/rested for each employee to the text file

For employees = 0 To noEmployees - 1

writeDaysWorkedToFile(totalNoShiftsRested, totalNoShiftsWorked, fileSaveLocation, employees)

totalNoShiftsRested(employees) = 0

totalNoShiftsWorked(employees) = 0

Next

End Sub

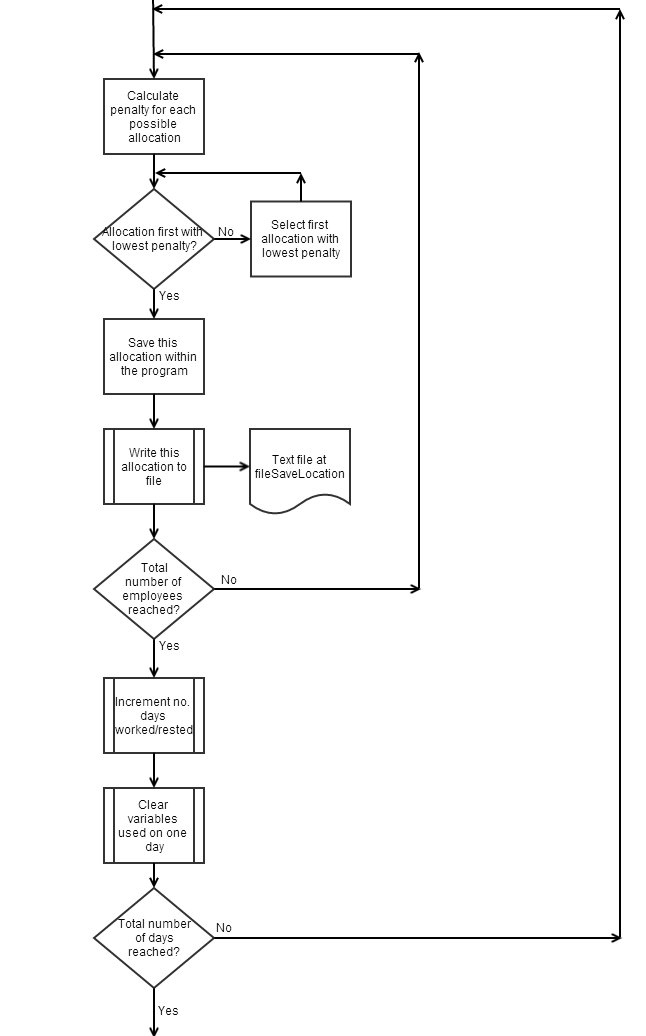
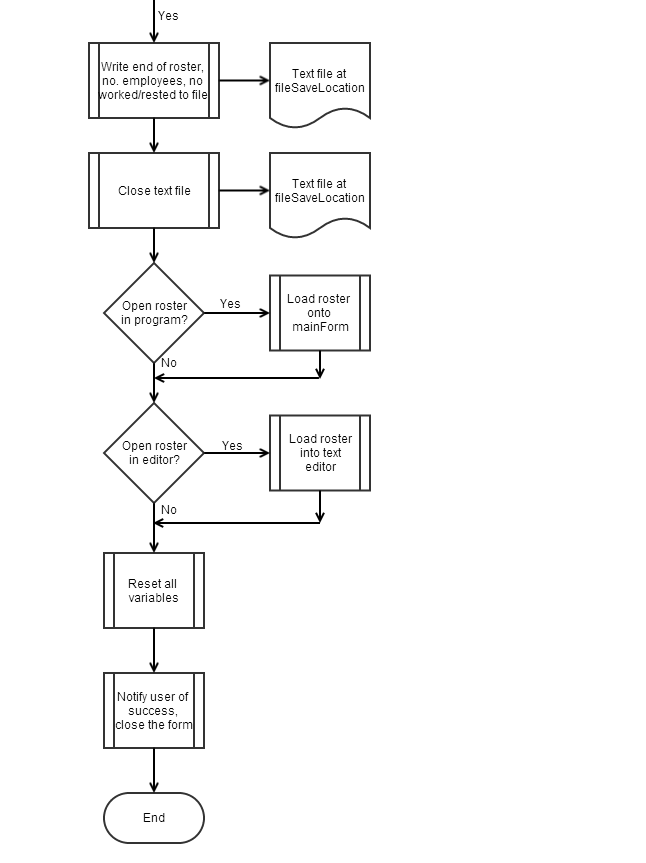
The majority of the processing (calculating penalties and finding the first allocation with lowest penalty) is done by the functions calculatePenalty and findAllocationLowestPenalty. The flowcharts for these algorithms are shown below, and their code can be found in appendix B (as with all the other code for the solution).

The sub also references built-in objects (StreamWriter) to write the end of roster and number of employees to the file. The days worked/rested values are written by a separate sub (writeDaysWorkedToFile) which also uses this technique.

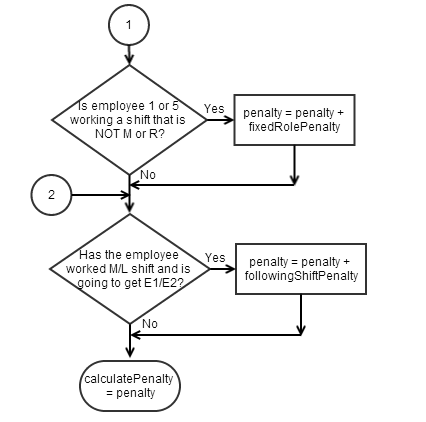
### C:\Users\James\Desktop\COLLEGE\Computing\COMP4\maintenance\roster_creation_flow_summary.pngAlgorithm design

General flowchart for roster creation: this does not show all processing.



********

### C:\Users\James\Desktop\COLLEGE\Computing\COMP4\maintenance\calculatepenalty.pngSpecific flowcharts for roster processing: calculatePenalty



See appendix B for a full listing of the code including this function (calculatePenalty).

### Specific flowcharts for roster processing: findAllocationLowestPenalty

****

See appendix B for a full listing of the code including this function (findAllocationLowestPenalty).

### Annotated program listing/code

Please see Appendix B for a complete listing of the commented code for the technical solution.

### System settings/Configuration

The system has been developed and tested on four primary machines, all running variants of the Windows operating system.

#### System 1

The system has primarily been developed on a Dell XPS computer running 64-bit Windows 7 Home Premium. This machine was also used to do the majority of testing for the system.

The IDE used was Microsoft Visual Basic 2010 Express edition. Microsoft Office applications, Outlook and Notepad++ were also used.

Specifications:

Intel Core i5-2430M at 2.4GHz

750GB SATA HDD

4GB RAM

1366x768 display resolution

#### System 2

The second system used during the development process was Knatt Consulting’s primary PC: a Toshiba laptop, also running 64-bit Windows 7 Home Premium. This has very similar specifications to the development machine and I am therefore reasonably confident that, upon installation, the system functions in the same way that it did on the development machine. Since I did not have access to the machine directly during the development phase of the system, I used email to transfer .zip files containing my solution to the client for him to test himself. This confirmed that there were little to no problems transferring the solution between computers.

Specifications:

Intel Core i5-2410M at 2.3GHz

500GB SATA HDD

4GB RAM

1366x768 display resolution

Outlook and Office installed

#### System 3

The third system used during the development process was located at college. Since the processor and overall specifications, including the operating system, were very similar, I used this machine to test whether the user interface was suitably easy to read on a higher resolution, larger screen. The solution adapted to the new screen size without any problems.

Specifications:

Intel Core i3 at 2.0GHz

250GB SATA HDD

4GB RAM

1600x1200 display resolution

#### System 4

The final system used during the development process was an older Toshiba laptop that was used to confirm whether the solution was too highly processor-intensive to run on older hardware. This system was also running an earlier OS – XP Home edition – which enabled me to test if the program ran stably on older releases of the operating system. As expected, the overall processing time for the solution was much higher, taking around three to four times as long for an average roster, but the program ran stably without ‘hanging’ or ‘crashing’. The smaller screen resolution did not affect the program.

Specifications:

Intel Pentium M-740 at 1.7GHz

512MB RAM

60GB ATA HDD

1280x800 screen resolution

### Research into changeover methods

#### Parallel changeover:

Old

New

The parallel changeover method involves running the old system alongside the new system.

The old system and new system are both used with new data.

This approach does invoke a higher cost, since both systems are run at the same time. The cost is both in money and time: staff have to do twice as much work, and they need to be paid for this. However, this is a low risk test since the old system can be reverted to reasonably easily. The new system is used for a documented length of time to allow for comparison, before the old system is phased out.

#### Direct changeover:

Old

New

A direct changeover method is used if the developer is confident that the system works – the old system is stopped and the new system is immediately started. This is a high risk changeover method since the systems are swapped instantly. However, it is a quick and direct changeover method since it goes straight into the new system. It assumes you’ve previously tested the system and fully trained the staff.

This changeover method is used for smaller systems where there isn’t going to be a high risk to the staff or company if there is a problem with the changeover. (This changeover method would be used for a self-employed person who owns a small shop, for example.)

#### Staggered/phased/staged changeover:

Old

New

Old

New

Old

New

A staggered changeover splits the process of changing the system over into individual changeover ‘tasks’ to introduce the staff to one new section at a time whilst still using the old. One example of this would be in a system for a small business that has multiple parts – the wage control system could be moved over to the new system whilst stock rotation is handled by the old system, for example.

In this changeover method, staff are gradually introduced to the whole system. This method is therefore lower risk, except for the fact that the individual parts in the system are still direct changeover.

#### Pilot changeover:

Old

New

The pilot changeover method is a prototype trial of the new system – this allows the user to give feedback on what needs to be changed. It is usually used in conjunction with one of the previous three methods after any issues with the new system found during this period have been corrected.

#### Retrospective parallel running:

New

Old

Retrospective parallel running is a changeover method that is similar to parallel running. Using this method, we run the new system with data from the old system, and the old system is kept running as if there had been no change. This enables us to test the new system with data for which we already know the result. (A worked example of this would involve the wages calculated using the old system for the previous month being compared to the results calculated by the new system working on the old data from last month.) This is likely to be one of the safest changeover methods; however, it is likely to incur significant time constraints, since it is likely that the new system needs to be run for a long time to test every part of it. However, this changeover method reduces training time for the system as the staff are able to learn on the job.

### Changeover method for the roster system

The changeover method which I chose for the implementation of the new rostering system was a direct changeover method. I chose this method because it allows a quick and simple change between systems, and I do not consider there to be a high risk to the company if there is a problem with the changeover. The system is to be installed on the client’s machine and will be introduced immediately, putting less strain on the company’s resources since the same amount of work will have to be done. To combat the problems that the user may have with adjusting to the new system, I intend to train the user to a certain level before the changeover, and I shall also provide documentation by way of a user guide, which will be accessible from the program interface.

# User Manual

In any completed system, a user guide is necessary in order for the users of the system to learn how to use it. For my system, I chose to produce a Word document that is incorporated within the program and can be accessed by selecting it from the ‘Help’ menu or pressing the F1 key, which will open it using the default program for Word .docx files.

For the completed user manual, please see Appendix D – user guide.

# Appraisal

### Comparison of solution to original objectives

At this stage in the project, evaluation of the solution I have created against the specified objectives that I set out at the start of the project allows me to see how suitable my solution is for the original problem. The objectives from the Analysis section of the project are re-printed below and coloured according to how well I feel my solution satisfies them.

Key:

Objective not met

Objective partially met

Objective met

|  |  |  |
| --- | --- | --- |
| **System objective** | | **Justification** |
| S1 | The program must provide a complete solution to the roster creation (it must create and display rosters). | The program both creates and displays rosters to the end user, using penalties to apply each constraint, and using data from the end user. Therefore, since it creates and displays rosters, this objective is met, and it is a ‘complete’ solution according to this specification point. |
| S2 | The system must allow a custom shift pattern to be entered in a form containing checkboxes to denote which shifts each employee can and cannot work. | Use of the form createCustomShifts1.vb (accessible from createRoster1.vb – the first form in the roster creation process) allows the user to define a shift pattern containing shifts that an employee can and cannot work. The custom shift pattern is made use of within the program and the penalty is incremented if the program has to assign a shift that is not permitted on this form. |
| S3 | All inputs will have validation to ensure that only correct and useful data is entered. The user will be prompted when the checks fail. | The program makes use of Regex and Try…Catch constructs on all applicable input fields to prevent invalid data entry ‘crashing’ the program. Prompts have been used to reiterate to the user what needs to be entered into each field on the form. |
| S4 | The system should allow a roster to be created for at least eight employees. | The Regex used within the program permits a roster that is between one and eight employees in length – theoretically the algorithm could be used for more employees, but it would require modification of Regex statements and the checkboxes on createCustomShifts1.vb. Therefore, I consider this specification point to be partially met. |
| S5 | The system should allow a roster to be created that is at least fifty days in length. | The Regex used within the program allows a roster to be created for up to ninety days in length, and therefore this objective has been met, since rosters that are at least fifty days in length can be created. |
| S6 | The system must display rosters in a grid-style output. | The system makes use of object-oriented programming to display a DataGridView control (using the ‘getters and setters’ in the module rosterGridView.vb). The roster is parsed from a specified file save location and is loaded into the grid. |
| S7 | The system should be able to email the created roster to each employee. | The system does not currently email created rosters to any of its users, therefore this is a specification point that has not currently been met, and therefore the program could be extended to allow this. |
| S8 | The system should display the roster in different ways for different viewers (depending on if they are employees/managers). | The system does not currently display personalized rosters for each employee according to the requirements set out in the Design section (employees do not need to see each others’ shifts, but a manager will need to see all of the shifts). |
| S9 | The completed installer and payloads for the program should take up less than 15MB of space on a memory stick. | The completed installer and payloads are under 1MB in size and therefore this objective has been met. |
| S10 | The complete system will be less than 20MB, including the documentation and any example rosters I choose to include. | The complete system when installed is around 1.2MB and is therefore less than 20MB, so this objective has been met. |
| S11 | The system must be able to be backed up onto a cloud-based storage system. | It is entirely possible to back the system up to the ‘cloud’ – it is recommended in the user guide documentation and maintenance sections of this project, and details have been provided as to how to do this. |
| S12 | The system must make use of standard Windows ‘Open’/’Save’ dialogs for the file I/O parts of the program. | The program uses the ‘Open’/’Save’ controls which have been adapted to replace existing files if necessary, and use Try…Catch constructs to ensure that the program does not crash if a file error is present. Therefore, this objective has been met. |
| S13 | The system must feature key shortcuts (eg F1) to allow common tasks in the program to be accessed. | The program makes use of the F1, F5, F6, F7 and ALT-F4 key combinations to access help, creating a new roster, loading a roster from file, and exiting the program respectively. Therefore, this objective has been met. |
| S14 | The system must not conflict with existing software on target machines when in use. | The end user reported that ‘the program…had no impact on any of the other functionality of [his] PC’, and during testing no conflicts were noted. Therefore, this objective has been met. |
| S15 | The system should feature a colour scheme that ties in with the Knatt Consulting house style. | The system does not currently feature a colour scheme that ties in with the Knatt Consulting house style; however, the installer does feature a Knatt Consulting logo and the icons used are similar in colour and style to this. |

|  |  |  |
| --- | --- | --- |
| **Processing objective** | | **Justification** |
| P1 | My system should process the data efficiently enough for it to create a roster in under two minutes when data is entered. | Rosters are typically able to be created in under ten seconds once the necessary data input has been made and the parameters selected. This means that the objective has been met. |
| P2 | My system’s data processing should be done so that it is error-free, producing rosters that exactly match the ones produced when the algorithm is traced by hand. | Identical rosters were produced using the hand trace method detailed in the Testing section (section 7) and the file contents check. Therefore this objective has been met, since the algorithm has been correctly implemented from the original pseudocode form. |
| P3 | The technique for processing of the rosters must include team days/public holidays. | Roster processing does not currently account for team days or public holidays, which was one feature considered at the start of the program. The system could be extended to account for this, but the rosters can still be manually edited to include team days and/or public holidays by the end user. |
| P4 | The system must process and export rosters as CSV files. | The system does export rosters into a fixed-width delimited format using slashes instead of commas, and thus rosters are exported as TXT files. Therefore, this objective has not been fully met. |
| P5 | The roster files must be formatted into a non-software specific format, that will allow them to be opened in a standard spreadsheet application. | Since the rosters are delimited using slashes, they can be loaded into a standard spreadsheet application, and can be subsequently processed if this is required. The format is one that I have created for this solution and is therefore ‘non-software specific’ – the rosters are not tied to a specific program. |
| P6 | The system must process and load created roster files into the grid in under twenty seconds on the target machine. | Rosters are typically loaded into the grid view almost instantly (with a ninety-day roster taking around three seconds maximum) and therefore this objective has been satisfied. |

|  |  |  |
| --- | --- | --- |
| **User objective** | | **Justification** |
| U1 | The user must be able to install the program from a memory stick in under five minutes on the target machine. | The installation process can be completed in under one minute since I have produced an easy-to-use installer using external software, which includes a default file path to save time upon installation. |
| U2 | Once installed, the user must be able to access the program from the Start menu and the Desktop. | The user is able to access the program from both of these places providing that they have selected these options within the installer. Therefore, this objective has been met. |
| U3 | Users should be asked to confirm whether they wish to exit before the program closes upon selecting ‘Exit’. | Users are prompted as shown in the User Manual section upon clicking ‘Exit’ or ‘Quit’ or selecting the red X in the corner of the window. |
| U4 | Users must be able to create a roster in under five minutes (including entry of parameters as detailed in the design section) | Users are able to create a roster in this time period: this was tested using the parameters specified in the Design section of the project, and therefore this objective has been satisfied. |
| U5 | The user must be able to enter penalties that are between 0 and at least 5000 for each rule. | The user is able to enter penalties that are between 0 and a maximum of 10000, which are validated by Regex to ensure that invalid data cannot be entered. |
| U6 | If the user enters a penalty of zero for a rule, that rule must not be applied to the roster creation. | Rules with penalties that are set to zero have no effect on the roster creation. Therefore, this objective is satisfied. |
| U7 | Users must be able to open more than one roster at a time in the program to allow comparison. | Users are not currently able to open more than one roster in the program at a time; however, they could run multiple instances of the program to get around this. As mentioned in the Maintenance section of the project, the program could be extended to allow this to happen by allowing multiple instances of the rosterGridView class to be created. |

### Deadlines

The solution was completed in late January 2014, meaning that it was installed in early February after beta testing on the end user’s machine and several other computers. The installation was completed with a direct changeover method, as discussed in the Systems Maintenance section. This meant that I was able to effectively satisfy the original deadline (Jan-Feb 2014) laid out in the Investigation and Analysis section.

### User feedback on the roster solution

As part of any completed project, the suitability of the solution created will vary depending on how well it meets the original criteria set out by the target user at the start of the project. The end user will have an opinion on the developed system, and it is important to take this into account. I chose to perform an interview as I felt that it would provide me with the best quality of feedback from my end user; however, I had to do this by email since Mr Knatt was out of the country for business, and it was difficult to arrange a suitable time for a video call due to time differences.

The questions that I compiled for the email interview took into account the following topics:

* General feedback about the system’s functionality
* Comparison of the system with the original objectives
* Ease of installation and use of the system
* Interface design feedback
* Constructive user criticism on certain parts of the system
* Suggestions about how they would like the system to be improved or extended

*Can I have some general feedback on the functionality of the program?*

The program has all the functionality that I requested and is straightforward and logical to use. Once installed, it opens as expected from the start bar or from the desktop icon. The program is driven by a series of 'wizard-like' screens which clearly indicate what inputs are needed at each stage of the build process. The outputs are easy to understand and can be imported onwards into a series of formats.

*How well do you feel that the program meets the original objectives?*

The program is a good replacement for the manual processes required to carry out the roster creation previously. It produces accurate repeatable results which have none of the minor errors that can be present in a complex manual process. The time taken to produce results is perfectly acceptable for the small and medium sized staff groups that I generally work with.

*How easy was the program to install/set up?*

The program was installed quickly and required virtually no configuration to set it up and get it up and running. I was particularly pleased to note that the set up and installation of the roster program had no impact on any of the other functionality of my PC, either during or after installation.

*Do you have any criticisms on the program or its output?*

One minor criticism is that the rules that I have inputted are not captured anywhere on the text output of the roster solution created. If I am running a number of options for comparison, it is useful to know which scenario relates to which solution. I believe this could be addressed in a future release.

*How functional is the program’s interface? What would you like to see improved?*

Having used the interface a number of times, I understand the way to start the roster creation process from the menu or function key options. However this could be made clear to a user on an opening screen with a text description or even a start button. As a proprietary product, I would also like the 'look and feel' of the interface to reflect my company image and house style as well as incorporating logos as appropriate.

*Do you have any suggestions as to how to expand on the functionality of the current program?*

The program is a good foundation which could be developed in a number of ways. In future, it could also be possible to have the whole functionality online to allow me to share and discuss the rostering process with others.

As a minimum, I would like the ability to share the roster solutions created online with clients for discussion and possible use by staff groups. At that stage, it would important to be able to filter the results to only show the roster for a particular staff member or sub-team.

### Analysis of user feedback

The feedback I received on my system was of a good enough depth to allow me to analyse it. Overall, Mr Knatt seems pleased with the system that I have produced and it solves the problem effectively: in his words, the program is ‘straightforward and logical to use’ and ‘easy to understand’.

Mr Knatt seems clear that the system meets the objectives that we set out at the start of the project – I was particularly pleased that he noted that the system is a ‘good replacement for the manual processes’ that were previously required, as one of these objectives was to remove the need for the subcontractor’s involvement in the roster creation process. He also notes that the time taken to produce rosters is perfectly acceptable, and the rosters (from his point of view) have no obvious errors, which was originally a problem discussed in the Analysis section of the project. The installation procedure – commonly a problem with new systems – seemed to go smoothly, requiring ‘virtually no configuration’ to set up and use, which was presumably partly due to the instructions in the user guide, although I did not specifically request feedback on this section. Therefore, from the feedback I have received on the processing and functionality of the system, I can conclude that the program meets the end user’s requirements.

In addition for asking for feedback on the parts of the system currently created, I also asked if Mr Knatt could provide feedback on possible areas of improvement for the system. The point that he makes about capturing the rules in the output of the program and displaying these within the grid view control is a very valid point, and as he states, it ‘could be addressed in a future release’ by writing the shift patterns, rules and penalties into the text file in the same way that the created shifts are written to the file. The program interface could also be improved – logos such as those featured in the installer could be easily placed on the forms of the program, and through additional coding, a start button/text description could be added, which would improve the functionality of the program and make it easier for new users to learn how to use the system.

Incorporating elements to allow the ‘functionality’ to be online would be much more costly in terms of time, but as Mr Knatt states, it would be a valuable asset to the company, allowing him to ‘share and discuss’ rosters. Online versions of the solution were discussed and these would need to have the features to ‘filter’ the roster as he stated – due to time constraints, I would have had difficulty during the implementation stage when it came to creating a way of viewing rosters according to specific employees: however, given more time, I believe that this would be possible.

In conclusion, I am satisfied with my user’s feedback on the solution: it raises several valid points about the functionality of the system, and confirms that I have produced a system that is successfully able to function within the workplace and will be genuinely useful to Knatt Consulting as a company. The core roster creation functionality is clearly present in the program and a future release of the program could address many of the points raised in the feedback that I received.

### Extensions

Extensions should be made to the future versions of the system to keep it relevant to the company’s changing problems as the roster creation process develops: months after the changeover is completed, the end user could still find additional problems that need addressing. In this section, I will specify extensions that I would make to any new releases of the system.

#### Modified interface design, featuring branding

Modifying the user interface to include logos and a clearer description when the program loads would improve the functionality of the program for a new user, as mentioned by Mr Knatt. The logos would also improve the look and feel of the created system – since it was created for a specific company, cohesive and consistent branding would improve the aesthetics of the company image.

The change would be made by incorporating embedded images into the forms that make up the program, and by creating a ‘loading screen’ that is displayed when the program first loads to allow quick access to common tasks, as is commonplace with commercial applications.

#### Rule capture in the roster text file

One of the key points brought up by the user feedback on the system was that the rules, penalties and shift patterns are not captured anywhere in the program’s text output. This means that comparison of the rosters is more difficult than it should be, since the consultant has to make extra notes (separate from the roster files themselves) as to which limitations were specified upon creation. Therefore, another possible extension of the system would be to have it write the penalty, days worked/rested and shift pattern values to the roster file. Modifying the code used to load the text file into the grid would also allow these values to be displayed within the program. Additional modification to allow the use of multiple instances of the rosterGridView object would also allow easier comparison of rosters, since the user would be able to compare the rosters directly within the program.

#### Online elements to the roster solution

One part of the system that was discussed in the Analysis and Design sections of the project was the possibility to have users being able to view the shift patterns they are to work, with a personalized shift pattern for each employee. This would require the program to make use of a web-based interface, which, due to time constraints, I would have struggled to implement before the deadlines set by my end user. However, it would be entirely possible to use JavaScript and HTML to parse a roster text file and load it into a web-page. User accounts could be created for each employee in the system and this would allow specific employees to view the shift patterns they are to work. Since Knatt Consulting has web-hosting space available on their server, this could be made use of, as discussed in the Design section.

Upon consideration of the idea of moving the whole system online, I have decided against this: although it would be possible, and would allow the system to be updated more easily (since only one version would require replacement if an update was required), it would require the replacement of the core VB elements of the system, and I am not confident that a JavaScript program could handle the data processing requirements within a browser-based scenario. Moving the display of rosters online is definitely an extension that could be made to the program, since the core roster creation elements would still be present on Mr Knatt’s computer.

# Appendices

* Appendix A: documents and interview transcription
* Appendix B: program listing
* Appendix C: test results
* Appendix D: user guide
* Appendix E: bibliography

# Appendix A – document analysis

This shows the documents mentioned in the analysis section.

### Document 1 – TRN Proposal 130513.docx



Knatt Consulting Ltd

The Oaks

Dunsfold Road

Plaistow

West Sussex

RH14 0PW

13th May 2013

Mr [name]

Operations Manager

[Address]

Dear [name],

**Roster Review and Amendments**

Following our kick-off meeting on 9th May at your offices, I have pleasure in confirming a proposal for consultancy support, working with you and your colleagues to achieve the business objectives discussed.

The following outlines my understanding of the scope needed to deliver key benefits for your business. I have broken the work into tasks which can be carried out independently or in parallel as appropriate.

**Task 1 – [Team A]**

This task will generate a modified roster for the [Team A] staff

Background

The roster for the small group of Revenue Centre staff requires updating and amendment following some changes to numbers and working practices made during late 2012. There is also an opportunity to change rosters for staff to reduce vacant lines in the current roster. This should ensure that staff are present in the right places and at the time to maximise revenue generated from ticket sales.

Proposed activities

* Initial meetings with management to review background and requirements in detail and collate relevant information
* Develop revised [Team A] roster (with options)
* Presentation and feedback of proposals and expected benefits

Proposed activities

* Meeting with management to understand detail of current roster patterns and rules and proposed changes
* Collate relevant information
* Meeting with your Learning and Development team to assess feasibility of different configurations of training days
* Develop revised roster proposals for [Teams B and C] staff (with options)
* Presentation and feedback of proposals and expected benefits

Duration and Costs

The following costs and durations are estimated for this task:

|  |  |  |  |
| --- | --- | --- | --- |
| **Task 2** | **[Teams B and C]** |  |  |
|  | Rate/ day | Days | Total (excl. VAT) |
| Principal Consultant | [£] | [] | [£] |
| Roster Analyst | [£] | [] | [£] |
|  |  |  | [£] |

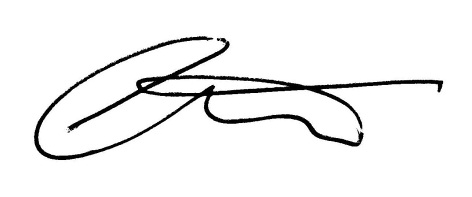
Based on our recent discussions, it is anticipated that this work will be completed by the end of June 2013.

All work will be carried out at your office locations and remotely in our offices as appropriate. It is assumed that full access will be provided to relevant personnel and information in electronic format to ensure efficient delivery of these tasks. We are, of course, happy to sign any appropriate confidentiality or non-disclosure agreement that your business may require.

Work will be delivered under the terms of our standard consultancy agreement or another similar contract form of your choice.

I trust this proposal now fully captures your requirements and can form a basis for us to proceed with the Task 1 and Task 2 work agreed.

Yours sincerely,



C.M. Knatt

Director – Knatt Consulting Ltd

Duration and Costs

The following costs and durations are estimated for this task:

|  |  |  |  |
| --- | --- | --- | --- |
| **Task 1** | **[Team A]** | |  |
|  | Rate/ day | Days | Total (excl. VAT) |
| Principal Consultant | [£] | [] | [] |
| Roster Analyst | [£] | [] | [] |
|  |  |  | [] |

Based on our current discussions it is expected that this task can be completed by 24th May.

**Task 2 – [Teams B and C]**

This task will be focussed on developing roster proposals for the above groups which incorporate changes to working practices which are in the process of being agreed as part of the recent pay round. Additionally, opportunities to improve the efficiency of training delivery will be investigated.

Background

As a result of the pay deal currently being concluded, a number of changes to staff working practices are anticipated. One example is a change to the arrangements for rostered training days which will result in an overall reduction of one day to be reallocated as a spare shift. At the same time, there is an opportunity to review the current mechanisms for delivering of training and develop rostering proposals which allow staff to be released in blocks of consecutive days rather than single days spread out over time.

Proposed activities

* Meeting with management to understand detail of current roster patterns and rules and proposed changes
* Collate relevant information
* Meeting with your Learning and Development team to assess feasibility of different configurations of training days
* Develop revised roster proposals for [Teams B and C] staff (with options)
* Presentation and feedback of proposals and expected benefits

Duration and Costs

The following costs and durations are estimated for this task:

|  |  |  |  |
| --- | --- | --- | --- |
| **Task 2** | **[Teams B and C]** |  |  |
|  | Rate/ day | Days | Total (excl. VAT) |
| Principal Consultant | [£] | [] | [£] |
| Roster Analyst | [£] | [] | [£] |
|  |  |  | [£] |

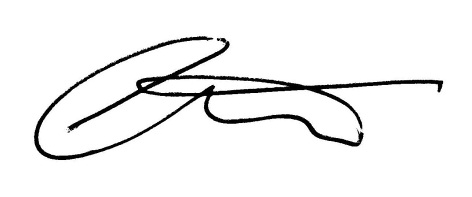
Based on our recent discussions, it is anticipated that this work will be completed by the end of June 2013.

All work will be carried out at your office locations and remotely in our offices as appropriate. It is assumed that full access will be provided to relevant personnel and information in electronic format to ensure efficient delivery of these tasks. We are, of course, happy to sign any appropriate confidentiality or non-disclosure agreement that your business may require.

Work will be delivered under the terms of our standard consultancy agreement or another similar contract form of your choice.

I trust this proposal now fully captures your requirements and can form a basis for us to proceed with the Task 1 and Task 2 work agreed.

Yours sincerely,



C.M. Knatt

Director – Knatt Consulting Ltd

**Document 2 – Roster (TRN Team A v01 280513.docx)**

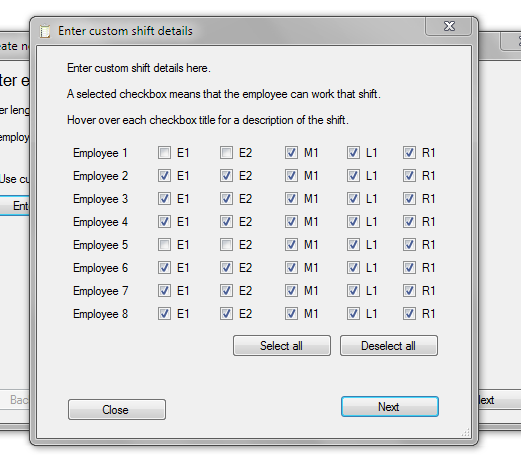
### Document 2 - TRN Team A v01 280513.xlsx

# Appendix B – program listing

This section contains the real code from a version of the roster solution. It is listed in the order that it appears in the Visual Basic project.

### createCustomShifts1.vb

createCustomShifts1.vb contains the code behind the form shown below. It allows the user to enter a custom shift pattern using checkbox elements.



Public Class createCustomShifts1

'this form is for the entry of a custom shift pattern

Private Sub selectAllButton\_Click(sender As System.Object, e As System.EventArgs) Handles selectAllButton.Click

'select all checkboxes and check them

For Each ctrl As Control In Me.Controls

If TypeOf ctrl Is CheckBox Then

DirectCast(ctrl, CheckBox).Checked = True

End If

Next

'disable the select all button and enable the deselect all button

selectAllButton.Enabled = False

deselectAllButton.Enabled = True

End Sub

Private Sub deselectAllButton\_Click(sender As System.Object, e As System.EventArgs) Handles deselectAllButton.Click

'select all checkboxes and uncheck them

For Each ctrl As Control In Me.Controls

If TypeOf ctrl Is CheckBox Then

DirectCast(ctrl, CheckBox).Checked = False

End If

Next

'disable the deselect all button and enable the select all button

deselectAllButton.Enabled = False

selectAllButton.Enabled = True

End Sub

Private Sub closeButton\_Click(sender As System.Object, e As System.EventArgs) Handles closeButton.Click

Call toExit() 'check if user wishes to quit

End Sub

Public Sub toExit()

Dim exitResponse As Integer

exitResponse = MsgBox("Are you sure that you wish to cancel entering custom shifts?", MsgBoxStyle.YesNo + vbQuestion, "Enter custom shifts")

If exitResponse = vbYes Then

Me.Close()

'stop custom shift pattern use

globalVariables.useCustomShifts = False

createRoster1.useCustomShiftsCheckBox.Checked = False

Else

End If

End Sub

Private Sub nextButton\_Click(sender As System.Object, e As System.EventArgs) Handles nextButton.Click

clearCustomShifts(globalVariables.customShifts) 'reset the custom shifts array each time the user saves a new set of shifts

Dim matches() As Control

For employees As Integer = 1 To 8

For j As Integer = 1 To 2

matches = Me.Controls.Find("emp" & employees & "E" & j & "CheckBox", True) 'this works for the E1 and E2 checkboxes

If matches.Length > 0 AndAlso TypeOf matches(0) Is CheckBox Then

Dim cb As CheckBox = DirectCast(matches(0), CheckBox)

If cb.Checked Then

globalVariables.customShifts(employees - 1, j - 1) = vbTrue 'set relevant array element to be true

End If

End If

Next

matches = Me.Controls.Find("emp" & employees & "M1CheckBox", True)

If matches.Length > 0 AndAlso TypeOf matches(0) Is CheckBox Then

Dim cb As CheckBox = DirectCast(matches(0), CheckBox)

If cb.Checked Then

globalVariables.customShifts(employees - 1, 2) = vbTrue

End If

End If

matches = Me.Controls.Find("emp" & employees & "L1CheckBox", True)

If matches.Length > 0 AndAlso TypeOf matches(0) Is CheckBox Then

Dim cb As CheckBox = DirectCast(matches(0), CheckBox)

If cb.Checked Then

globalVariables.customShifts(employees - 1, 3) = vbTrue

End If

End If

matches = Me.Controls.Find("emp" & employees & "R1CheckBox", True)

If matches.Length > 0 AndAlso TypeOf matches(0) Is CheckBox Then

Dim cb As CheckBox = DirectCast(matches(0), CheckBox)

If cb.Checked Then

globalVariables.customShifts(employees - 1, 4) = vbTrue

End If

End If

Next

MsgBox("Your custom shift settings have been saved.", vbInformation, "Create new roster")

globalVariables.useCustomShifts = True

Me.Close()

End Sub

Sub clearCustomShifts(customShifts(,) As Boolean)

For i = 0 To 7

For j = 0 To 4

customShifts(i, j) = vbFalse 'replace every element in the array with a False boolean value

Next

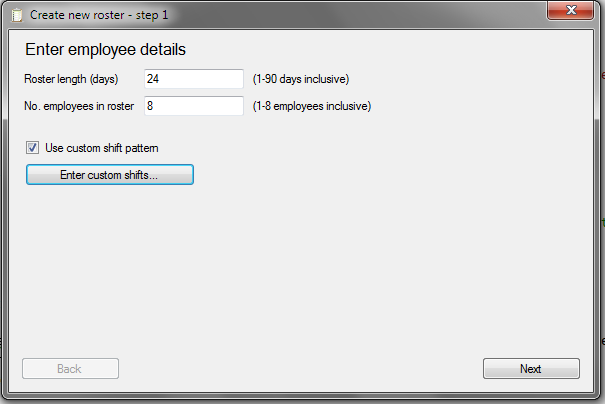
Next

End Sub

End Class

### createRoster1.vb

createRoster1.vb contains the code behind the first stage of a ‘wizard-style’ form that the user uses as part of the roster creation process. It also loads the previously mentioned form ‘createCustomShifts1.vb’ if the custom shifts checkbox is selected and the button clicked.



Imports System.Text.RegularExpressions 'needed for the regex validation

Public Class createRoster1

Dim valid As Boolean

Private Sub createRoster1\_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

End Sub

Public Sub nextButton1\_Click(sender As System.Object, e As System.EventArgs) Handles nextButton1.Click

validateEntry(valid)

If valid = True Then 'only allow the user to move on if the input is valid

nextForm() 'calls sub to change form

End If

End Sub

Function validateEntry(ByRef valid As Boolean)

valid = False

Dim errorMessageDisplayed As Boolean

Dim dayRegex As Regex = New Regex("^(90|[1-8][0-9]|[1-9])$") 'matches 1-90 inclusive

Dim empRegex As Regex = New Regex("^([1-8])$") 'matches 1-8 inclusive

Dim dayMatch, empMatch As Match

'try...catch to catch any overflows or accidental text entry

Try

globalVariables.days = noDaysEntry.Text

globalVariables.employees = noEmployeesEntry.Text

Catch ex As Exception

MsgBox("Please enter an integer day value between 1 and 90 inclusive, and an employee value between 1 and 8 inclusive." & \_

vbCrLf & ex.Message, MsgBoxStyle.Exclamation, "Create new roster")

errorMessageDisplayed = True

valid = False

End Try

'regex match to check if the numbers are within set boundaries

dayMatch = dayRegex.Match(globalVariables.days)

empMatch = empRegex.Match(globalVariables.employees)

If empMatch.Success = True And dayMatch.success = True Then

valid = True

Else

valid = False

If errorMessageDisplayed = False Then

MsgBox("Please enter an integer day value between 1 and 90 inclusive, and an employee value between 1 and 8 inclusive.", \_

MsgBoxStyle.Exclamation, "Create new roster")

End If

End If

Return valid

End Function

Sub nextForm()

Me.Close()

createRoster2.Show()

End Sub

Private Sub customShiftsEntryButton\_Click(sender As System.Object, e As System.EventArgs) Handles customShiftsEntryButton.Click

'create a new instance of the custom shifts form

Dim createCustomShiftsForm As New createCustomShifts1

createCustomShiftsForm.ShowDialog()

End Sub

Private Sub useCustomShiftsCheckBox\_CheckedChanged(sender As System.Object, e As System.EventArgs) Handles useCustomShiftsCheckBox.CheckedChanged

'if the entry button should be enabled and it isn't, enable it

'if the entry button shouldn' be enabled and it is, disable it

If customShiftsEntryButton.Enabled = False Then

customShiftsEntryButton.Enabled = True

ElseIf customShiftsEntryButton.Enabled = True Then

customShiftsEntryButton.Enabled = False

End If

If useCustomShiftsCheckBox.Checked = True Then

globalVariables.useCustomShifts = True

Else

globalVariables.useCustomShifts = False

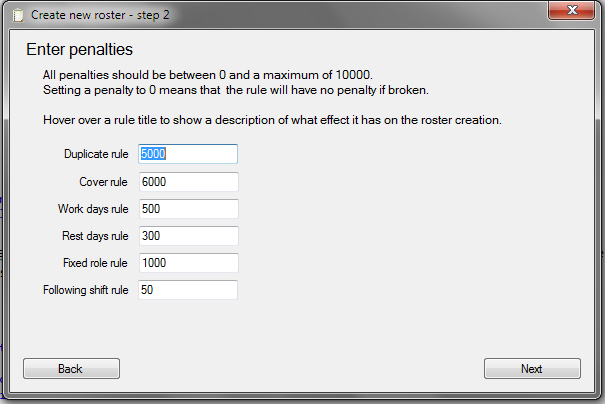
End If

End Sub

End Class

### createCustomShifts2.vb

createCustomShifts2.vb contains the code behind the second ‘wizard screen’ where the user enters details about the penalty for each rule.



Imports System.Text.RegularExpressions 'required for regex validation

Public Class createRoster2

Dim valid As Boolean

Sub nextButton1\_Click(sender As System.Object, e As System.EventArgs) Handles nextButton1.Click

validateEntry(valid) 'check if valid before allowing the user to move on

If valid = True Then

nextForm()

End If

End Sub

Private Sub backButton1\_Click(sender As System.Object, e As System.EventArgs) Handles backButton1.Click

prevForm()

End Sub

Function validateEntry(ByRef valid As Boolean)

Dim errorMessageDisplayed As Boolean

Dim penaltyRegex As Regex = New Regex("^(10000|[0-9]{0,4})$") 'matches 0-10000 inclusive

Dim penaltyMatch As Match

valid = True

'try...catch to catch any overflows or accidental text entry

Try

penalties(0) = duplicatePenaltyEntry.Text

penalties(1) = coverPenaltyEntry.Text

penalties(2) = workDaysPenaltyEntry.Text

penalties(3) = restDaysPenaltyEntry.Text

penalties(4) = fixedRolePenaltyEntry.Text

penalties(5) = followingShiftPenaltyEntry.Text

Catch ex As Exception

MsgBox("Please ensure all penalties have integer values between 0 and 10000 inclusive." & \_

vbCrLf & ex.Message, MsgBoxStyle.Exclamation, "Create new roster")

errorMessageDisplayed = True

valid = False

End Try

For count = 0 To 5

penaltyMatch = penaltyRegex.Match((penalties(count)).ToString)

'if any element in the array fails the regex test then the whole validation check fails

If penaltyMatch.Success = False Then

valid = False

End If

Next

If valid = False And errorMessageDisplayed = False Then

MsgBox("Please ensure all penalties have integer values between 0 and 10000 inclusive.", \_

MsgBoxStyle.Exclamation, "Create new roster")

End If

Return valid

End Function

Sub nextForm()

Me.Close()

createRoster3.Show()

End Sub

Sub prevForm()

Me.Close()

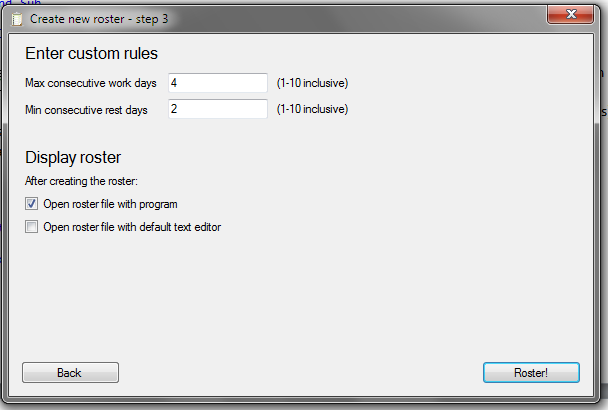
createRoster1.Show()

End Sub

End Class

### createRoster3.vb

createRoster3.vb contains the code that creates and saves the roster, using the values entered on the previous two forms (stored in the global variables file), and the values entered on this form. Upon validation of the values on this form using Regex notation (similar to the previous two forms), the user is prompted for a save location and the new roster is written to a text file in this location with a user-specified file name.



Imports System.Text.RegularExpressions 'used for regex validation

Public Class createRoster3

#Region "Variables"

'dim form inputs

Dim noDays, noEmployees, duplicatePenalty, coverPenalty, workDaysPenalty, restDaysPenalty, \_

fixedRolePenalty, followingShiftPenalty As Integer

Dim fileSaveLocation As String 'used by BIF savedialog

Dim maxConsecutiveWork, minConsecutiveRest As Integer

Dim valid, quit As Boolean

'dim roster basics

Dim possibleAllocations() As String = {"E1", "E2", "M", "L", "R"}

Dim noShiftsPrevAllocated(,) As Object = {{"E1", 0}, {"E2", 0}, {"M", 0}, {"L", 0}, {"R", 0}}

Dim prevDayAllocation(7) As String

Dim noConsecutiveDaysWorked() As Integer = {"0", "0", "0", "0", "0", "0", "0", "0"}

Dim noConsecutiveDaysRested() As Integer = {"0", "0", "0", "0", "0", "0", "0", "0"}

Dim totalNoShiftsWorked(7) As Integer

Dim totalNoShiftsRested(7) As Integer

Dim shiftPosition As Integer

Dim maxNoEmployees As Integer

Dim penalty As Integer

Dim shiftAllocated As String

Dim minPenalty As Integer

Dim allocationMinPenalty(,) As Object = {{"", ""}}

Dim allocationsPenalties(,) As Object = {{"E1", 0}, {"E2", 0}, {"M", 0}, {"L", 0}, {"R", 0}}

#End Region

Sub validateEntry(ByRef valid As Boolean)

Dim errorMessageDisplayed As Boolean

Dim workRestRegex As Regex = New Regex("^(10|[1-9])$") 'matches 1-10 inclusive

Dim workMatch As Match

Dim restMatch As Match

'try...catch to catch any overflows or accidental text entry

Try

maxConsecutiveWork = maxConsecutiveWorkEntry.Text

minConsecutiveRest = minConsecutiveRestEntry.Text

Catch ex As Exception

MsgBox("Max consecutive work days and min consecutive rest days must be integer values between 1 and 10 inclusive." & \_

vbCrLf & ex.Message, MsgBoxStyle.Exclamation, "Create new roster")

errorMessageDisplayed = True

valid = False

End Try

workMatch = workRestRegex.Match(maxConsecutiveWork.ToString)

restMatch = workRestRegex.Match(minConsecutiveRest.ToString)

If workMatch.Success = True And restMatch.Success = True Then 'both need to have passed the regex validation before the program can move on

valid = True

Else : valid = False

End If

If valid = False And errorMessageDisplayed = False Then

MsgBox("Max consecutive work days and min consecutive rest days must be integer values between 1 and 10 inclusive.", \_

MsgBoxStyle.Exclamation, "Create new roster")

End If

End Sub

Public Sub createRosterButton\_Click(sender As System.Object, e As System.EventArgs) Handles createRosterButton.Click

validateEntry(valid) 'validate final entry

If valid = True Then

processRoster()

End If

End Sub

Sub processRoster()

'previously validated variables are taken from the globalVariables file and saved for use here

coverPenalty = globalVariables.penalties(0)

duplicatePenalty = globalVariables.penalties(1)

workDaysPenalty = globalVariables.penalties(2)

restDaysPenalty = globalVariables.penalties(3)

fixedRolePenalty = globalVariables.penalties(4)

followingShiftPenalty = globalVariables.penalties(5)

noEmployees = globalVariables.employees

noDays = globalVariables.days

maxNoEmployees = noEmployees - 1

saveRoster(fileSaveLocation, quit) 'prompt user to select a save location BEFORE the processing is done

If quit = False Then

createNewRoster() 'calls algorithm

MsgBox("Creation successful! Roster saved to " & fileSaveLocation, MsgBoxStyle.Information, "Create new roster")

Me.Close()

If displayInProgramCheckBox.Checked = True Then

globalVariables.loadSilently = True

mainForm.loadRoster(fileSaveLocation, globalVariables.loadSilently) 'loads roster from a known location without any dialog boxes

End If

If displayInEditorCheckBox.Checked = True Then 'user wishes to load the file into a text editor

openRoster(fileSaveLocation)

End If

clearAll() 'reset all variables

Else

clearAll()

Me.Close()

End If

End Sub

Public Sub createNewRoster()

'for each day in the roster

For days = 1 To noDays

'for each employee on the day

For employees = 0 To noEmployees - 1

'calculate penalty for each possible allocation

For count = 0 To 4

allocationsPenalties(count, 1) = calculatePenalty(penalty, employees, count)

Next

'find first allocation with lowest penalty

findAllocationLowestPenalty(allocationsPenalties, minPenalty, allocationMinPenalty)

'set the allocated shift to be the allocation with minimum penalty

shiftAllocated = allocationMinPenalty(0, 0)

'if a work shift is allocated, the number of work days is increased and the number of rest days reset

'if a rest shift is allocated, the number of rest days is increased and the number of work days reset

If shiftAllocated <> "R" Then

noConsecutiveDaysWorked(employees) = noConsecutiveDaysWorked(employees) + 1

noConsecutiveDaysRested(employees) = 0

Else

noConsecutiveDaysRested(employees) = noConsecutiveDaysRested(employees) + 1

noConsecutiveDaysWorked(employees) = 0

End If

'save the new allocation (shiftAllocated) into shiftPrevAllocated (shift allocated to preceding employees

'ON THE CURRENT DAY) so that it is not assigned again unless needed

shiftPosition = Array.IndexOf(possibleAllocations, shiftAllocated) 'finds position of the previously allocated shift

noShiftsPrevAllocated(shiftPosition, 1) = noShiftsPrevAllocated(shiftPosition, 1) + 1

'save shiftAllocated into prevDayAllocation(employee)

prevDayAllocation(employees) = shiftAllocated

'write the shift allocation for this employee to file

writeAllocationsToFile(days, employees, allocationMinPenalty, fileSaveLocation)

Next 'next employee

'reset the variables which are only used on one day

shiftAllocated = ""

noShiftsPrevAllocated = {{"E1", 0}, {"E2", 0}, {"M", 0}, {"L", 0}, {"R", 0}}

For employees = 0 To noEmployees - 1

If prevDayAllocation(employees) = "R" Then

totalNoShiftsRested(employees) = totalNoShiftsRested(employees) + 1

Else

totalNoShiftsWorked(employees) = totalNoShiftsWorked(employees) + 1

End If

Next

Next 'next day

'write end of roster to file

Using writer As New IO.StreamWriter(fileSaveLocation, True)

writer.WriteLine("&&&")

End Using

'write no of employees to file

Using writer As New IO.StreamWriter(fileSaveLocation, True)

writer.WriteLine(noEmployees)

End Using

'write total number of days worked/rested for each employee to the text file

For employees = 0 To noEmployees - 1

writeDaysWorkedToFile(totalNoShiftsRested, totalNoShiftsWorked, fileSaveLocation, employees)

totalNoShiftsRested(employees) = 0

totalNoShiftsWorked(employees) = 0

Next

'job done

End Sub

Function calculatePenalty(ByVal penalty As Integer, ByVal employees As Integer, ByVal count As Integer)

penalty = 0

'duplicate rule:

'if a previously assigned work shift is allocated when it has already been allocated then the penalty is increased

If possibleAllocations(count) <> "R" And (noShiftsPrevAllocated(count, 1) <> 0) Then

penalty = penalty + duplicatePenalty

'ElseIf ((noShiftsPrevAllocated(count, 1) > 4) Or (noShiftsPrevAllocated(count, 1) > 4)) And possibleAllocations(count) = "R" Then

' penalty = penalty + duplicatePenalty

Else

penalty = penalty + 0

End If

'cover rule:

'if a rest shift is going to be allocated but this would mean that not all work shifts are covered, then the penalty is increased

'(ie if the total number of employees minus the number of work shifts previously assigned is less than the number of work shifts NOT allocated yet)

If possibleAllocations(count) = "R" And (maxNoEmployees - (noShiftsPrevAllocated(0, 1) + noShiftsPrevAllocated(1, 1) + \_

noShiftsPrevAllocated(2, 1) + noShiftsPrevAllocated(3, 1) + noShiftsPrevAllocated(4, 1)) < \_

(4 - (noShiftsPrevAllocated(0, 1) + noShiftsPrevAllocated(1, 1) + noShiftsPrevAllocated(2, 1) + noShiftsPrevAllocated(3, 1)))) Then

penalty = penalty + coverPenalty

End If

'work days rule

'if a work shift is assigned but the number of consecutive days worked is greater than the maximum, then the penalty is increased

If (noConsecutiveDaysWorked(employees) >= maxConsecutiveWork) And possibleAllocations(count) <> "R" Then

penalty = penalty + workDaysPenalty

End If

'rest days rule

'if a work shift is assigned but the employee has not previously been assigned the minimum number of rest days, then the penalty is increased

If (noConsecutiveDaysRested(employees) < minConsecutiveRest And (noConsecutiveDaysRested(employees) <> 0)) And possibleAllocations(count) <> "R" Then

penalty = penalty + restDaysPenalty

End If

'fixed role rule

'If the user has not entered custom shifts, employee 1 and 5 are only allowed to work shifts M and R. If they are assigned to another shift, the penalty is increased.

If globalVariables.useCustomShifts = False Then

If employees = 0 And possibleAllocations(count) <> "M" And possibleAllocations(count) <> "R" Then

penalty = penalty + fixedRolePenalty

ElseIf employees = 4 And possibleAllocations(count) <> "M" And possibleAllocations(count) <> "R" Then

penalty = penalty + fixedRolePenalty

End If

'If the user has entered custom shifts, and the program has to assign a shift that is not permitted by the user (ie permitted=false), the penalty is increased.

ElseIf globalVariables.useCustomShifts = True Then 'user has entered custom shifts

If possibleAllocations(count) = "E1" And customShifts(employees, 0) = False Then

penalty = penalty + fixedRolePenalty

ElseIf possibleAllocations(count) = "E2" And customShifts(employees, 1) = False Then

penalty = penalty + fixedRolePenalty

ElseIf possibleAllocations(count) = "M" And customShifts(employees, 2) = False Then

penalty = penalty + fixedRolePenalty

ElseIf possibleAllocations(count) = "L" And customShifts(employees, 3) = False Then

penalty = penalty + fixedRolePenalty

ElseIf possibleAllocations(count) = "R" And customShifts(employees, 4) = False Then

penalty = penalty + fixedRolePenalty

End If

End If

'following shift rule

'if employee has previously worked either an M or L shift and is going to be assigned to an E1 or an E2, the penalty is increased

If (prevDayAllocation(employees) = ("M")) And ((possibleAllocations(count) = ("E1")) Or (possibleAllocations(count) = ("E2"))) \_

Or (prevDayAllocation(employees) = ("L")) And ((possibleAllocations(count) = ("E1")) Or (possibleAllocations(count) = ("E2")) Or (possibleAllocations(count) = "M")) Then

penalty = penalty + followingShiftPenalty

Else

penalty = penalty + 0

End If

calculatePenalty = penalty

End Function

Function findAllocationLowestPenalty(ByVal allocationsPenalties(,) As Object, ByRef minPenalty As Integer, ByRef allocationMinPenalty(,) As Object)

minPenalty = 900000

Dim finalcount As Integer

allocationMinPenalty = {{""}, {""}}

Dim count As Integer = 0

For count = 0 To 4

'if the allocation's penalty is less than the minimum penalty, then replace the minimum penalty with this value

If allocationsPenalties(count, 1) < minPenalty Then minPenalty = allocationsPenalties(count, 1)

Next

For count = 0 To 4

'find the first allocation with this minimum penalty and exit upon finding it

If allocationsPenalties(count, 1) = minPenalty Then

finalcount = count

Exit For

End If

Next

allocationMinPenalty = {{(allocationsPenalties(finalcount, 0)), (allocationsPenalties(finalcount, 1))}}

findAllocationLowestPenalty = allocationMinPenalty 'returns allocation with the minimum penalty

End Function

Sub writeAllocationsToFile(days As Integer, employees As Integer, ByVal allocationMinPenalty(,) As Object, \_

fileSaveLocation As String)

Dim daysToWrite As String = ""

Dim penaltyToWrite As String = ""

Dim shiftToWrite As String = ""

'ensures that the day number written to file is always a two digit number

If (days <= 9 And days > 0) Then daysToWrite = "0" & days

If days > 9 Then daysToWrite = days

'ensures that the penalty number written to file is always a four digit number

If allocationMinPenalty(0, 1).ToString.Length = 1 Then

penaltyToWrite = "000" & allocationMinPenalty(0, 1)

ElseIf allocationMinPenalty(0, 1).ToString.Length = 2 Then

penaltyToWrite = "00" & allocationMinPenalty(0, 1)

ElseIf allocationMinPenalty(0, 1).ToString.Length = 3 Then

penaltyToWrite = "0" & allocationMinPenalty(0, 1)

Else : penaltyToWrite = allocationMinPenalty(0, 1)

End If

'if the shift saved does not have a 1 on the end, eg R, then add it

If allocationMinPenalty(0, 0).ToString.Length = 1 Then

shiftToWrite = allocationMinPenalty(0, 0) & "1"

Else : shiftToWrite = allocationMinPenalty(0, 0)

End If

'use an OOP form to write the allocation, penalty, days, shift to file

Using writer As New IO.StreamWriter(fileSaveLocation, True)

writer.Write(daysToWrite & "/" & (employees + 1) & "/" & \_

shiftToWrite & "/" & penaltyToWrite & "/")

End Using

End Sub

Sub writeDaysWorkedToFile(totalNoShiftsRested, totalNoShiftsWorked, fileSaveLocation, employees)

'use an OOP form to write the total no. days worked/rested to file

Using writer As New IO.StreamWriter(fileSaveLocation, True)

writer.WriteLine("Employee: " & employees + 1 & " Total worked/rested: " & totalNoShiftsWorked(employees) & "/" & \_

totalNoShiftsRested(employees) & " ")

End Using

End Sub

Public Sub clearAll() 'resets all variables used in roster creation

noDays = 0

noEmployees = 0

duplicatePenalty = 0

coverPenalty = 0

workDaysPenalty = 0

restDaysPenalty = 0

fixedRolePenalty = 0

followingShiftPenalty = 0

shiftPosition = 0

minPenalty = 0

maxNoEmployees = 0

penalty = 0

maxConsecutiveWork = 0

minConsecutiveRest = 0

possibleAllocations(0) = "E1"

possibleAllocations(1) = "E2"

possibleAllocations(2) = "M"

possibleAllocations(3) = "L"

possibleAllocations(4) = "R"

shiftAllocated = ""

fileSaveLocation = "" 'used by savedialog

allocationMinPenalty(0, 0) = {{"", ""}}

allocationsPenalties(0, 0) = {"E1", "0"}

allocationsPenalties(0, 0) = {"E2", "0"}

allocationsPenalties(0, 0) = {"M", "0"}

allocationsPenalties(0, 0) = {"L", "0"}

allocationsPenalties(0, 0) = {"R", "0"}

noShiftsPrevAllocated(0, 0) = {"E1", 0}

noShiftsPrevAllocated(0, 0) = {"E2", 0}

noShiftsPrevAllocated(0, 0) = {"M", 0}

noShiftsPrevAllocated(0, 0) = {"L", 0}

noShiftsPrevAllocated(0, 0) = {"R", 0}

For count = 0 To 7 'loop through each variable in these arrays and set it to zero

prevDayAllocation(count) = ""

noConsecutiveDaysWorked(count) = "0"

noConsecutiveDaysRested(count) = "0"

totalNoShiftsWorked(count) = 0

totalNoShiftsRested(count) = 0

Next

End Sub

Public Function saveRoster(ByRef fileSaveLocation As String, ByRef quit As Boolean) As Object

'uses saveRosterDialog object on the form to create a validated save dialog box

Dim response, cancelRoster As Integer

saveRosterDialog.Title = "Save as a TXT file"

saveRosterDialog.Filter = "TXT Files|\*.txt"

response = saveRosterDialog.ShowDialog()

If response = Windows.Forms.DialogResult.Cancel Then 'user has cancelled roster creation

cancelRoster = MsgBox("Are you sure you want to cancel creating the roster?", vbYesNo + vbQuestion, "Create new roster")

If cancelRoster = vbYes Then

quit = True 'quit to main form

Else : saveRoster(fileSaveLocation, quit)

End If

ElseIf response = Windows.Forms.DialogResult.OK Then

Try

fileSaveLocation = saveRosterDialog.FileName

Catch ex As Exception

MsgBox("Unable to write to file, or an unspecified error occurred" & vbCrLf & ex.Message, vbCritical, "Create new roster")

End Try

End If

'if the user selects 'replace' when saving the file, the old file needs to be overwritten, not amended.

If My.Computer.FileSystem.FileExists(fileSaveLocation) Then

My.Computer.FileSystem.DeleteFile(fileSaveLocation)

End If

saveRoster = fileSaveLocation

End Function

Sub openRoster(ByRef filesavelocation As String)

Try

If System.IO.File.Exists(filesavelocation) = True Then

Process.Start(filesavelocation) 'checks to see if file is available and opens it with the defaul text editing program

End If

Catch ex As Exception

MsgBox("Error: system could not find the file saved to. File cannot be opened." & vbCrLf & ex.Message, vbExclamation, "Create new roster")

End Try

End Sub

Private Sub backButton2\_Click(sender As System.Object, e As System.EventArgs) Handles backButton2.Click

Me.Close() 'user needs to return to the previous form

createRoster2.Show()

End Sub

End Class

### globalVariables.vb

globalVariables.vb is a short Visual Basic module file that I have used to enable variables that need to be accessed by different forms to be passed across the forms.

Module globalVariables

'variable declarations to enable variables to be passed across forms

Public days As Integer

Public employees As Integer

Public penalties(5) As Integer

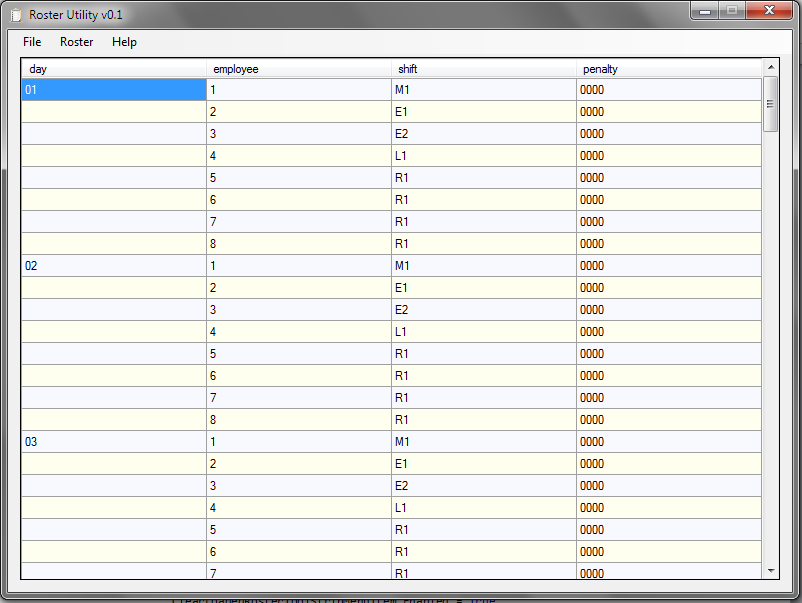
Public customShifts(7, 4) As Boolean

Public useCustomShifts, loadSilently As Boolean

End Module

### mainForm.vb

mainForm.vb contains the code behind the first form that runs when the user opens the program. From here, the roster creation, viewing utility and documentation are visible using a dropdown ‘toolStripMenu’. The code that loads an existing roster is also part of this form.



Public Class mainForm

Dim splitAtSlashes As String()

Dim fileSaveLocation As String

Dim rosterLoaded As Boolean

Private Sub mainForm\_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

changeDropDown(rosterLoaded) 'used to check if a roster is loaded and change the drop down toolbar item appropriately

End Sub

Sub changeDropDown(ByVal rosterLoaded)

'if roster is loaded, you can clear it, however, you are not able to clear it if it is not loaded

If rosterLoaded = True Then

ClearLoadedRosterToolStripMenuItem.Enabled = True

Else : ClearLoadedRosterToolStripMenuItem.Enabled = False

End If

End Sub

Public Sub toExit()

Dim exitResponse As Integer

exitResponse = MsgBox("Are you sure that you wish to exit?", MsgBoxStyle.YesNo + MsgBoxStyle.Question, "Roster Utility v0.1")

If exitResponse = vbYes Then 'user wishes to exit program

Me.Close()

Else

End If

End Sub

Public Sub dataGridRosterView\_MouseUp(ByVal sender As System.Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles dataGridRosterView.MouseDown

'handles a user clicking on the datagridview

'check to see if user has right clicked

If e.Button = Windows.Forms.MouseButtons.Right Then

'create a new context menu

Dim cms = New ContextMenuStrip

Dim item1 = cms.Items.Add("Clear loaded roster")

item1.Tag = 1

If rosterLoaded = True Then

item1.Enabled = True 'since the roster can only be cleared if one is loaded

Else : item1.Enabled = False

End If

AddHandler item1.Click, AddressOf menuChoice

Dim item2 = cms.Items.Add("Load new roster")

item2.Tag = 2

AddHandler item2.Click, AddressOf menuChoice

'show the context menu to the user

cms.Show(dataGridRosterView, e.Location)

End If

End Sub

Private Sub menuChoice(ByVal sender As Object, ByVal e As EventArgs)

Dim item = CType(sender, ToolStripMenuItem)

Dim selection = CInt(item.Tag)

If selection = 1 Then

clearRoster(rosterLoaded) 'if clear roster selected, run sub to clear the datagridview

ElseIf selection = 2 Then

loadRoster(fileSaveLocation, globalVariables.loadSilently) 'loadSilently will be set to false here since a file save location needs to be specified

End If

End Sub

#Region "load roster from file into display"

Sub loadRoster(ByRef fileSaveLocation As String, ByVal loadSilently As Boolean)

loadRosterDialog.Title = "Open a supported TXT file"

loadRosterDialog.Filter = "TXT Files|\*.txt"

Dim response, cancelRoster As Integer

If loadSilently = False Then 'save location needs to be found using the save dialog box

response = loadRosterDialog.ShowDialog()

If response = Windows.Forms.DialogResult.OK Then

Try 'helps to prevent the program crashing if an illegal file name is entered

fileSaveLocation = loadRosterDialog.FileName

If My.Computer.FileSystem.FileExists(fileSaveLocation) Then

readTxtRoster(fileSaveLocation, rosterLoaded)

'passes the file name to the sub that reads the roster

Else

MsgBox("The system could not find the text file to read.", MessageBoxButtons.OK & MessageBoxIcon.Error, "Load roster from file")

End If

Catch ex As Exception

MsgBox("The system could not find the text file to read." & vbCrLf & ex.Message, MessageBoxButtons.OK & MessageBoxIcon.Error, "Load roster from file")

End Try

ElseIf response = Windows.Forms.DialogResult.Cancel Then 'user wishes to cancel loading the roster

Do

cancelRoster = MsgBox("Are you sure you wish to cancel loading a roster?", vbYesNo + MsgBoxStyle.Question, "Load roster from file")

If cancelRoster = vbNo Then

loadRosterDialog.ShowDialog() 'reopen the load dialog

rosterLoaded = False 'ensure the program knows no roster is loaded

changeDropDown(rosterLoaded) 'change the menu items accordingly

End If

Loop Until cancelRoster = vbYes 'user is 100% sure that they wish to quit

End If

ElseIf loadSilently = True Then

'this part of the program is used if the 'display within program' check box is checked on createRoster3

'since the user does not specify a file save location, it has previously been saved within the program

Try

'ensures the file saved correctly

If My.Computer.FileSystem.FileExists(fileSaveLocation) Then

readTxtRoster(fileSaveLocation, rosterLoaded)

'passes the file name to the sub that reads the roster

Else

MsgBox("The system could not find the text file to read.", MessageBoxButtons.OK & MessageBoxIcon.Error, "Load roster from file")

End If

Catch ex As Exception

MsgBox("The system could not find the text file to read." & vbCrLf & ex.Message, MessageBoxButtons.OK & MessageBoxIcon.Error, "Load roster from file")

End Try

End If

End Sub

Sub readTxtRoster(ByVal fileSaveLocation As String, ByRef rosterLoaded As Boolean)

Dim firstLineOfRoster As String

Dim numberOfEmployeesInRoster As String

MsgBox("Loading text roster from " & fileSaveLocation, vbInformation, "Load roster from file")

Using r As System.IO.StreamReader = New System.IO.StreamReader(fileSaveLocation)

firstLineOfRoster = r.ReadLine 'the roster is written to the first line of the file

numberOfEmployeesInRoster = r.ReadLine 'the number of employees in the roster is written to the second

End Using

splitRoster(firstLineOfRoster, splitAtSlashes) 'the string that is read has to be split up at the slash delimiters

fillDataGridView(splitAtSlashes, numberOfEmployeesInRoster) 'load the roster into the datagridview

rosterLoaded = True 'if successful, the roster has been loaded, and the dropdown menu is changed accordingly

changeDropDown(rosterLoaded)

End Sub

Sub splitRoster(ByVal firstLineOfRoster, ByRef splitAtSlashes)

'this sub splits the first line of the roster at the slashes by defining a slash as the new delimiting char

'the string is split from 01/1/M1/0000/...... to 01 1 M1 0000 and each element is saved into an array

splitAtSlashes = firstLineOfRoster.Split(New Char() {"/"c})

End Sub

Sub fillDataGridView(ByVal splitAtSlashes, numberOfEmployeesInRoster)

Dim list = New List(Of rosterGridView)

Dim j As Integer = 0

For i = 0 To (splitAtSlashes.length) - 2

If j Mod 8 = 0 Then

'j mod 8 = 0 when we reach the first employee on each day of the roster

'write the day as well as the other information to a new row in the datagridview

list.Add(New rosterGridView(splitAtSlashes(i), splitAtSlashes(i + 1), splitAtSlashes(i + 2), splitAtSlashes(i + 3))) 'adds a new row to the gridview with day

Else

'the day has previously been written so it does not need to be written again

list.Add(New rosterGridView("", splitAtSlashes(i + 1), splitAtSlashes(i + 2), splitAtSlashes(i + 3)))

End If

If j <> numberOfEmployeesInRoster - 1 Then

j = j + 1 'j loops through each employee in the roster until it gets to the end and then it is reset to 0

Else : j = 0

End If

i = i + 3

Next

dataGridRosterView.DataSource = list

End Sub

Sub clearRoster(ByRef rosterLoaded)

Dim clearResponse As Integer = MsgBox("Are you sure that you want to clear the loaded roster?", MsgBoxStyle.Question + MsgBoxStyle.YesNo, "Clear loaded roster?")

If clearResponse = vbYes Then

dataGridRosterView.DataSource = Nothing 'reset the datagridview

rosterLoaded = False 'permit user to load a new roster

Else

rosterLoaded = True

End If

changeDropDown(rosterLoaded) 'change the dropdown menu accordingly

End Sub

#End Region

#Region "menuStripItems"

'handles the user clicking on the menu items in the menu strip

Private Sub ExToolStripMenuItem\_Click(sender As System.Object, e As System.EventArgs) Handles ExToolStripMenuItem.Click

Call toExit()

End Sub

Private Sub CreateNewRosterToolStripMenuItem\_Click(sender As System.Object, e As System.EventArgs) Handles CreateNewRosterToolStripMenuItem.Click

createRoster1.Show()

End Sub

Private Sub loadExistingRosterToolStripMenuItem\_Click(sender As System.Object, e As System.EventArgs) Handles loadExistingRosterToolStripMenuItem.Click

Call loadRoster(fileSaveLocation, globalVariables.loadSilently)

End Sub

Private Sub ClearLoadedRosterToolStripMenuItem\_Click(sender As System.Object, e As System.EventArgs) Handles ClearLoadedRosterToolStripMenuItem.Click

Call clearRoster(rosterLoaded)

End Sub

#Region "documentation"

Private Sub ViewDocumentationToolStripMenuItem\_Click(sender As System.Object, e As System.EventArgs) Handles ViewDocumentationToolStripMenuItem.Click

Try 'prevents the program crashing if the documentation cannot be found

System.Diagnostics.Process.Start("Example documentation for the roster solution.docx")

Catch ex As Exception

MsgBox("Failed to load file. " & vbCrLf & ex.Message, vbCritical, "Roster Utility v0.1")

End Try

End Sub

Private Sub KnattConsultingOnTheWebToolStripMenuItem\_Click(sender As System.Object, e As System.EventArgs) Handles KnattConsultingOnTheWebToolStripMenuItem.Click

Try 'prevents the program crashing if the web page cannot be loaded

System.Diagnostics.Process.Start("http://www.knattconsulting.com/")

Catch ex As Exception

MsgBox("Failed to load web page. " & vbCrLf & ex.Message, vbCritical, "Roster Utility v0.1")

End Try

End Sub

#End Region

#End Region

End Class

### rosterGridView.vb

rosterGridView.vb is a module that contains the ‘getters and setters’ behind the dataGridView which is embedded on mainForm.vb. The dataGridView is used to display the roster to the user.

Public Class rosterGridView

Public Sub New(ByVal day As String, ByVal employee As String, ByVal shift As String, ByVal penalty As String)

'creates a new object with these properties, used to load things into the dgv

\_day = day

\_employee = employee

\_shift = shift

\_penalty = penalty

With mainForm.dataGridRosterView

'alternates the datagrid colour styles to make it easier to read

.RowsDefaultCellStyle.BackColor = Color.GhostWhite

.AlternatingRowsDefaultCellStyle.BackColor = Color.Ivory

End With

End Sub

Private \_day As String

Public Property day() As String

Get

Return \_day

End Get

Set(ByVal value As String)

\_day = value

End Set

End Property

Private \_employee As String

Public Property employee() As String

Get

Return \_employee

End Get

Set(ByVal value As String)

\_employee = value

End Set

End Property

Private \_shift As String

Public Property shift() As String

Get

Return \_shift

End Get

Set(ByVal value As String)

\_shift = value

End Set

End Property

Private \_penalty As String

Public Property penalty() As String

Get

Return \_penalty

End Get

Set(ByVal value As String)

\_penalty = value

End Set

End Property

End Class

# Appendix C – Test data

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Test purpose** | **Screenshot** | |
| 1.1a | Validate number of days. | Entering typical data allows the user to move onto createRoster2.vb since validation checks are passed. | |
| 1.1b | Validate number of days. | Entering extreme data allows the user to move onto createRoster2.vb since validation checks are passed. | |
| 1.1c | Validate number of days. |  | |
| 1.2a | Validate no. of employees | Entering typical data allows the user to move onto createRoster2.vb since validation checks are passed. | |
| 1.2b | Validate no. of employees | Entering extreme data allows the user to move onto createRoster2.vb since validation checks are passed. | |
| 1.2c | Validate no. of employees |  | |
| 2.1a | Validate duplicate penalty entry. | Entering typical data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.1b | Validate duplicate penalty entry. | Entering extreme data allows the user to move onto createRoster2.vb since validation checks are passed. | |
| 2.1c | Validate duplicate penalty entry. |  | |
| 2.2a | Validate cover penalty entry. | Entering typical data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.2b | Validate cover penalty entry. | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.2c | Validate cover penalty entry. | Here, we can see that a decimal penalty has been accepted, allowing the user to move onto createRoster3.vb when this should not be permitted. Corrective action for this error in the program required modification of the regex statement used to validate the input boxes. | |
| 2.3a | Validate work days penalty entry. | Entering typical data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.3b | Validate work days penalty entry. | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.3c | Validate work days penalty entry. |  | |
| 2.4a | Validate rest days penalty entry. | Entering typical data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.4b | Validate rest days penalty entry. | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.4c | Validate rest days penalty entry. |  | |
| 2.5a | Validate fixed role penalty entry. | Entering typical data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.5b | Validate fixed role penalty entry. | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.5c | Validate fixed role penalty entry. | Here, we can see that a decimal penalty has been accepted, allowing the user to move onto createRoster3.vb when this should not be permitted. Corrective action for this error in the program required modification of the regex statement used to validate the input boxes. | |
| 2.6a | Validate following shift penalty entry. | Entering typical data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.6b | Validate following shift penalty entry. | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 2.6c | Validate following shift penalty entry. |  | |
| 3.1a | Validate max consecutive work days. | Entering typical data allows the save dialog to be loaded because the validation checks are passed. | |
| 3.1b | Validate max consecutive work days. | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 3.1c | Validate max consecutive work days. |  | |
| 3.2a | Validate min consecutive rest days | Entering typical data allows the save dialog to be loaded because the validation checks are passed. | |
| 3.2b | Validate min consecutive rest days | Entering extreme data allows the user to move onto createRoster3.vb since validation checks are passed. | |
| 3.2c | Validate min consecutive rest days |  | |
| **No** | **Test purpose** |  |
| 4.1a | Check roster clearing. |  |
| 4.1b | Check roster clearing. |  |
| 5.1a | Check flow of control through forms. (1) |  |
| 5.1b | Check flow of control through forms. (1) |  |
| 5.2a | Check flow of control through forms. (2) |  |
| 5.2b | Check flow of control through forms. (2) |  |
| 5.3a | Check flow of control through forms. (3) |  |
| 5.3b | Check flow of control through forms. (3) |  |
| 5.4a | Check flow of control through forms. (4) |  |
| 5.4b | Check flow of control through forms. (4) |  |
| 6.1 | Menu testing - quit |  |
| 6.2 | Menu testing – create new roster |  |
| 6.3 | Menu testing – load roster (1) |  |
| 6.4a | Menu testing – load roster (2) |  |
| 6.4b | Menu testing – load roster (2) |  |
| 6.5 | Menu testing – clear roster (1) |  |
| 6.6 | Menu testing – clear roster (2) |  |
| 6.7 | Menu testing – ‘help’ (1) |  |
| 6.8 | Menu testing – ‘help’ (2) |  |

### Section 7 – hand traced algorithm design

This section contains a version of the Excel tables used to run an initial trace of the brute force algorithm pseudocode in the Design section of the project. I used this to test if the algorithm produced acceptable results by running the algorithm by hand in Excel, and comparing it to the output from versions of the coded program. The table produced the same results as the program, so the tests were successful.

#### Constraints used

The constraints listed below and their respective penalties were applied to each employee and each of their possible allocations for each day. This is identical to the method used in early versions of the solution. (Later versions included an extra penalty set for the custom shift pattern, which affects the following shift rule).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hard constraints** | **Test no. of penalty points if violated** |  |  |  | | |  | |  |  |  |
| Daily cover rule | 5000 |  | Shift | No. of staff req'd: | | |  | |  |  |  |
|  |  |  | E1 | 1 | | |  | |  |  |  |
|  |  |  | E2 | 1 | | |  | |  |  |  |
|  |  |  | M | 1 | | |  | |  |  |  |
|  |  |  | L | 1 | | |  | |  |  |  |
|  |  |  | R | 4 | | |  | |  |  |  |
|  |  |  |  |  | | |  | |  |  |  |
| **Soft constraints** | **Test no. of penalty points if violated** |  |  |  | | |  | |  |  |  |
| Work days rule | 500 |  | Max no of consecutive work days |  | | |  | |  |  |  |
|  |  |  | 4 |  | | |  | |  |  |  |
|  |  |  |  |  | | |  | |  |  |  |
| Rest days rule |  |  | Min no of consecutive rest days |  | | |  | |  |  |  |
|  | 300 |  | 2 |  | | |  | |  |  |  |
|  |  |  |  |  | | |  | |  |  |  |
|  |  |  | Staff member | Possible allocations: | | | | |  |  |  |
| Fixed role rule | 250 |  | Leader A | M | R | | | |  |  |  |
|  |  |  | Leader B | M | R | | | |  |  |  |
|  |  |  |  |  |  | | | |  |  |  |
|  |  |  | Previous shift | Can only be followed by: | | | | |  | |  |
| Following shift rule: | 50 |  | E1 | E1 | | E2 | | M | L | | R |
|  |  |  | E2 | E2 | | E1 | | M | L | | R |
|  |  |  | M | M | | L | | R |  | |  |
|  |  |  | L | L | | R | |  |  | |  |
|  |  |  | R | R | | E1 | | E2 | M | | L |

The below table was filled out for each employee and allocation on each day. I calculated the penalty by hand and selected the first allocation with the minimum penalty to be each employee’s final allocation, as the algorithm in the program does.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Staff/ day | Possible alloc’n: | Penalty | Violation | Possible alloc’n: | Penalty | Violation | Possible alloc’n: | Penalty | Violation | Possible alloc’n: | Penalty | Violation | Possible alloc’n: | Penalty | Violation | **First min penalty alloc’n:** |
| 11 | E1 | 250 | Fixed role | E2 | 250 | Fixed role | M | 0 | None | L | 250 | Fixed role | R | 0 | None | **M** |
| 21 | E1 | 0 | None | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E1** |
| 31 | E1 | 5000 | Cover | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E2** |
| 41 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 0 | None | R | 0 | None | **L** |
| 51 | E1 | 5250 | Cover, fixed role | E2 | 5250 | Cover, fixed role | M | 5000 | Cover | L | 5250 | Cover, fixed role | R | 0 | None | **R** |
| 61 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 71 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 81 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 12 | E1 | 300 | Fixed role, following shift | E2 | 300 | Fixed role, following shift | M | 0 | None | L | 250 | Fixed role | R | 0 | None | **M** |
| 22 | E1 | 0 | None | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E1** |
| 32 | E1 | 5000 | Cover | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E2** |
| 42 | E1 | 5050 | Cover, following shift | E2 | 5050 | Cover, following shift | M | 5050 | Cover, following shift | L | 0 | None | R | 0 | None | **L** |
| 52 | E1 | 5550 | Cover, min rest days, fixed role | E2 | 5550 | Cover, min rest days, fixed role | M | 5300 | Cover, min rest days | L | 5550 | Cover, min rest days, fixed role | R | 0 | None | **R** |
| 62 | E1 | 5300 | Cover, min rest days | E2 | 5300 | Cover, min rest days | M | 5300 | Cover, min rest days | L | 5300 | Cover, min rest days | R | 0 | None | **R** |
| 72 | E1 | 5300 | Cover, min rest days | E2 | 5300 | Cover, min rest days | M | 5300 | Cover, min rest days | L | 5300 | Cover, min rest days | R | 0 | None | **R** |
| 82 | E1 | 5300 | Cover, min rest days | E2 | 5300 | Cover, min rest days | M | 5300 | Cover, min rest days | L | 5300 | Cover, min rest days | R | 0 | None | **R** |
| 13 | E1 | 300 | Fixed role, following shift | E2 | 300 | Fixed role, following shift | M | 0 | None | L | 250 | Fixed role | R | 0 | None | **M** |
| 23 | E1 | 0 | None | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E1** |
| 33 | E1 | 5000 | Cover | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E2** |
| 43 | E1 | 5050 | Cover, following shift | E2 | 5050 | Cover, following shift | M | 5050 | Cover, following shift | L | 0 | None | R | 0 | None | **L** |
| 53 | E1 | 5250 | Cover, fixed role | E2 | 5250 | Cover, fixed role | M | 5000 | Cover | L | 5250 | Cover, fixed role | R | 0 | None | **R** |
| 63 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 73 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 83 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 14 | E1 | 300 | Fixed role, following shift | E2 | 300 | Fixed role, following shift | M | 0 | None | L | 250 | Fixed role | R | 0 | None | **M** |
| 24 | E1 | 0 | None | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E1** |
| 34 | E1 | 5050 | Cover, following shift | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 0 | None | **E2** |
| 44 | E1 | 5050 | Cover, following shift | E2 | 5050 | Cover, following shift | M | 5050 | Cover, following shift | L | 0 | None | R | 0 | None | **L** |
| 54 | E1 | 5250 | Cover, fixed role | E2 | 5250 | Cover, fixed role | M | 5000 | Cover | L | 5000 | Cover, fixed role | R | 0 | None | **R** |
| 64 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 74 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 84 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 5000 | Cover | R | 0 | None | **R** |
| 15 | E1 | 800 | Work days, fixed role, following shift | E2 | 800 | Work days, fixed role, following shift | M | 500 | Work days | L | 750 | Work days, fixed role | R | 0 | None | **R** |
| 25 | E1 | 500 | Work days | E2 | 500 | Work days | M | 500 | Work days | L | 500 | Work days | R | 0 | None | **R** |
| 35 | E1 | 550 | Work days, following shift | E2 | 550 | Work days, following shift | M | 500 | Work days | L | 500 | Work days | R | 0 | None | **R** |
| 45 | E1 | 550 | Work days, following shift | E2 | 550 | Work days, following shift | M | 550 | Work days, following shift | L | 500 | Work days | R | 0 | None | **R** |
| 55 | E1 | 250 | Fixed role | E2 | 250 | Fixed role | M | 0 | None | L | 250 | Fixed role | R | 5000 | Cover | **M** |
| 65 | E1 | 0 | None | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 5000 | Cover | **E1** |
| 75 | E1 | 5000 | Cover | E2 | 0 | None | M | 5000 | Cover | L | 0 | None | R | 5000 | Cover | **E2** |
| 85 | E1 | 5000 | Cover | E2 | 5000 | Cover | M | 5000 | Cover | L | 0 | None | R | 5000 | Cover | **L** |
| 16 | E1 | 600 | Rest days, fixed role, following shift | E2 | 600 | Rest days, fixed role, following shift | M | 300 | Rest days | L | 550 | Rest days, fixed role | R | 0 | None | **R** |
| 26 | E1 | 300 | Rest days | E2 | 300 | Rest days | M | 300 | Rest days | L | 300 | Rest days | R | 0 | None | **R** |
| 36 | E1 | 300 | Rest days | E2 |  | Rest days | M | 300 | Rest days | L | 300 | Rest days | R | 0 | None | **R** |
| 46 | E1 | 300 | Rest days | E2 |  | Rest days | M | 300 | Rest days | L | 300 | Rest days | R | 0 | None | **R** |
| 56 | E1 | 300 | Fixed role, following shift | E2 | 300 | Fixed role, following shift | M | 0 | None | L | 250 | Fixed role | R | 5000 | Cover | **M** |
| 66 | E1 | 0 | None | E2 | 0 | None | M | 5000 | Cover | L | 50 | Following shift | R | 5000 | Cover | **E1** |
| 76 | E1 | 5000 | Cover | E2 | 0 | None | M | 5050 | Cover, following shift | L | 50 | Following shift | R | 5000 | Cover | **E2** |
| 86 | E1 | 5050 | Cover, following shift | E2 | 5050 | Cover, following shift | M | 5050 | Cover, following shift | L | 0 | None | R | 5000 | Cover | **L** |

This table shows the final work allocation for each employee on each day, for the first six days of the roster. Creating this meant that I was able to compare it with the output from the program and see if the created rosters were the same, which they were. This meant that the testing was a success, and the program passed this test, as you can see from the right hand table, which shows the roster that was created by the program. (The labels of the shifts in the second version have changed slightly to ensure that the output was of a consistent number of characters.)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Day |  |  |  |  |  |
| Staff member |  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Leader A | M | M | M | M | R | R |
| 2 | Staff A1 | E1 | E1 | E1 | E1 | R | R |
| 3 | Staff A2 | E2 | E2 | E2 | E2 | R | R |
| 4 | Staff A3 | L | L | L | L | R | R |
| 5 | Leader B | R | R | R | R | M | M |
| 6 | Staff B1 | R | R | R | R | E1 | E1 |
| 7 | Staff B2 | R | R | R | R | E2 | E2 |
| 8 | Staff B3 | R | R | R | R | L | L |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Staff member | Day 1 | Day 2 | Day 3 | Day 4 | Day 5 | Day 6 |
| 1 | M1 | M1 | M1 | M1 | R1 | R1 |
| 2 | E1 | E1 | E1 | E1 | R1 | R1 |
| 3 | E2 | E2 | E2 | E2 | R1 | R1 |
| 4 | L1 | L1 | L1 | L1 | R1 | R1 |
| 5 | R1 | R1 | R1 | R1 | M1 | M1 |
| 6 | R1 | R1 | R1 | R1 | E1 | E1 |
| 7 | R1 | R1 | R1 | R1 | E2 | E2 |
| 8 | R1 | R1 | R1 | R1 | L1 | L1 |

|  |  |  |
| --- | --- | --- |
| **No** | **Test purpose** | **Evidence** |
| 8.1a | File check (1) |  |
| 8.1b | File check (1) |  |
| 8.2a | File check (2) |  |
| 8.2b | File check (2) |  |
| 8.3a | File contents check | Contents of txtRoster1.txt: 01/1/M1/0000/01/2/E1/0000/01/3/E2/0000/01/4/L1/0000/01/5/R1/0000/01/6/R1/0000/01/7/R1/0000/01/8/R1/0000/02/1/M1/0000/02/2/E1/0000/02/3/E2/0000/02/4/L1/0000/02/5/R1/0000/02/6/R1/0000/02/7/R1/0000/02/8/R1/0000/03/1/M1/0000/03/2/E1/0000/03/3/E2/0000/03/4/L1/0000/03/5/R1/0000/03/6/R1/0000/03/7/R1/0000/03/8/R1/0000/04/1/M1/0000/04/2/E1/0000/04/3/E2/0000/04/4/L1/0000/04/5/R1/0000/04/6/R1/0000/04/7/R1/0000/04/8/R1/0000/05/1/R1/0000/05/2/R1/0000/05/3/R1/0000/05/4/R1/0000/05/5/M1/0000/05/6/E1/0000/05/7/E2/0000/05/8/L1/0000/06/1/R1/0000/06/2/R1/0000/06/3/R1/0000/06/4/R1/0000/06/5/M1/0000/06/6/E1/0000/06/7/E2/0000/06/8/L1/0000/07/1/M1/0000/07/2/E1/0000/07/3/E2/0000/07/4/L1/0000/07/5/R1/0000/07/6/R1/0000/07/7/R1/0000/07/8/R1/0000/08/1/M1/0000/08/2/E1/0000/08/3/E2/0000/08/4/L1/0000/08/5/R1/0000/08/6/R1/0000/08/7/R1/0000/08/8/R1/0000/09/1/M1/0000/09/2/E1/0000/09/3/E2/0000/09/4/L1/0000/09/5/R1/0000/09/6/R1/0000/09/7/R1/0000/09/8/R1/0000/10/1/M1/0000/10/2/E1/0000/10/3/E2/0000/10/4/L1/0000/10/5/R1/0000/10/6/R1/0000/10/7/R1/0000/10/8/R1/0000/11/1/R1/0000/11/2/R1/0000/11/3/R1/0000/11/4/R1/0000/11/5/M1/0000/11/6/E1/0000/11/7/E2/0000/11/8/L1/0000/12/1/R1/0000/12/2/R1/0000/12/3/R1/0000/12/4/R1/0000/12/5/M1/0000/12/6/E1/0000/12/7/E2/0000/12/8/L1/0000/13/1/M1/0000/13/2/E1/0000/13/3/E2/0000/13/4/L1/0000/13/5/R1/0000/13/6/R1/0000/13/7/R1/0000/13/8/R1/0000/14/1/M1/0000/14/2/E1/0000/14/3/E2/0000/14/4/L1/0000/14/5/R1/0000/14/6/R1/0000/14/7/R1/0000/14/8/R1/0000/15/1/M1/0000/15/2/E1/0000/15/3/E2/0000/15/4/L1/0000/15/5/R1/0000/15/6/R1/0000/15/7/R1/0000/15/8/R1/0000/16/1/M1/0000/16/2/E1/0000/16/3/E2/0000/16/4/L1/0000/16/5/R1/0000/16/6/R1/0000/16/7/R1/0000/16/8/R1/0000/17/1/R1/0000/17/2/R1/0000/17/3/R1/0000/17/4/R1/0000/17/5/M1/0000/17/6/E1/0000/17/7/E2/0000/17/8/L1/0000/18/1/R1/0000/18/2/R1/0000/18/3/R1/0000/18/4/R1/0000/18/5/M1/0000/18/6/E1/0000/18/7/E2/0000/18/8/L1/0000/19/1/M1/0000/19/2/E1/0000/19/3/E2/0000/19/4/L1/0000/19/5/R1/0000/19/6/R1/0000/19/7/R1/0000/19/8/R1/0000/20/1/M1/0000/20/2/E1/0000/20/3/E2/0000/20/4/L1/0000/20/5/R1/0000/20/6/R1/0000/20/7/R1/0000/20/8/R1/0000/21/1/M1/0000/21/2/E1/0000/21/3/E2/0000/21/4/L1/0000/21/5/R1/0000/21/6/R1/0000/21/7/R1/0000/21/8/R1/0000/22/1/M1/0000/22/2/E1/0000/22/3/E2/0000/22/4/L1/0000/22/5/R1/0000/22/6/R1/0000/22/7/R1/0000/22/8/R1/0000/23/1/R1/0000/23/2/R1/0000/23/3/R1/0000/23/4/R1/0000/23/5/M1/0000/23/6/E1/0000/23/7/E2/0000/23/8/L1/0000/24/1/R1/0000/24/2/R1/0000/24/3/R1/0000/24/4/R1/0000/24/5/M1/0000/24/6/E1/0000/24/7/E2/0000/24/8/L1/0000/&&&  8  Employee: 1 Total worked/rested: 16/8  Employee: 2 Total worked/rested: 16/8  Employee: 3 Total worked/rested: 16/8  Employee: 4 Total worked/rested: 16/8  Employee: 5 Total worked/rested: 8/16  Employee: 6 Total worked/rested: 8/16  Employee: 7 Total worked/rested: 8/16  Employee: 8 Total worked/rested: 8/16 |
| 8.3b | File contents check | Contents of txtRoster2.txt:  01/1/M1/0000/01/2/E1/0000/01/3/E2/0000/01/4/L1/0000/01/5/R1/0000/01/6/R1/0000/01/7/R1/0000/01/8/R1/0000/02/1/M1/0000/02/2/E1/0000/02/3/E2/0000/02/4/L1/0000/02/5/R1/0000/02/6/R1/0000/02/7/R1/0000/02/8/R1/0000/03/1/M1/0000/03/2/E1/0000/03/3/E2/0000/03/4/L1/0000/03/5/R1/0000/03/6/R1/0000/03/7/R1/0000/03/8/R1/0000/04/1/M1/0000/04/2/E1/0000/04/3/E2/0000/04/4/L1/0000/04/5/R1/0000/04/6/R1/0000/04/7/R1/0000/04/8/R1/0000/05/1/R1/0000/05/2/R1/0000/05/3/R1/0000/05/4/R1/0000/05/5/M1/0000/05/6/E1/0000/05/7/E2/0000/05/8/L1/0000/06/1/R1/0000/06/2/R1/0000/06/3/R1/0000/06/4/R1/0000/06/5/M1/0000/06/6/E1/0000/06/7/E2/0000/06/8/L1/0000/07/1/M1/0000/07/2/E1/0000/07/3/E2/0000/07/4/L1/0000/07/5/R1/0000/07/6/R1/0000/07/7/R1/0000/07/8/R1/0000/08/1/M1/0000/08/2/E1/0000/08/3/E2/0000/08/4/L1/0000/08/5/R1/0000/08/6/R1/0000/08/7/R1/0000/08/8/R1/0000/09/1/M1/0000/09/2/E1/0000/09/3/E2/0000/09/4/L1/0000/09/5/R1/0000/09/6/R1/0000/09/7/R1/0000/09/8/R1/0000/10/1/M1/0000/10/2/E1/0000/10/3/E2/0000/10/4/L1/0000/10/5/R1/0000/10/6/R1/0000/10/7/R1/0000/10/8/R1/0000/11/1/R1/0000/11/2/R1/0000/11/3/R1/0000/11/4/R1/0000/11/5/M1/0000/11/6/E1/0000/11/7/E2/0000/11/8/L1/0000/12/1/R1/0000/12/2/R1/0000/12/3/R1/0000/12/4/R1/0000/12/5/M1/0000/12/6/E1/0000/12/7/E2/0000/12/8/L1/0000/13/1/M1/0000/13/2/E1/0000/13/3/E2/0000/13/4/L1/0000/13/5/R1/0000/13/6/R1/0000/13/7/R1/0000/13/8/R1/0000/14/1/M1/0000/14/2/E1/0000/14/3/E2/0000/14/4/L1/0000/14/5/R1/0000/14/6/R1/0000/14/7/R1/0000/14/8/R1/0000/15/1/M1/0000/15/2/E1/0000/15/3/E2/0000/15/4/L1/0000/15/5/R1/0000/15/6/R1/0000/15/7/R1/0000/15/8/R1/0000/16/1/M1/0000/16/2/E1/0000/16/3/E2/0000/16/4/L1/0000/16/5/R1/0000/16/6/R1/0000/16/7/R1/0000/16/8/R1/0000/17/1/R1/0000/17/2/R1/0000/17/3/R1/0000/17/4/R1/0000/17/5/M1/0000/17/6/E1/0000/17/7/E2/0000/17/8/L1/0000/18/1/R1/0000/18/2/R1/0000/18/3/R1/0000/18/4/R1/0000/18/5/M1/0000/18/6/E1/0000/18/7/E2/0000/18/8/L1/0000/19/1/M1/0000/19/2/E1/0000/19/3/E2/0000/19/4/L1/0000/19/5/R1/0000/19/6/R1/0000/19/7/R1/0000/19/8/R1/0000/20/1/M1/0000/20/2/E1/0000/20/3/E2/0000/20/4/L1/0000/20/5/R1/0000/20/6/R1/0000/20/7/R1/0000/20/8/R1/0000/21/1/M1/0000/21/2/E1/0000/21/3/E2/0000/21/4/L1/0000/21/5/R1/0000/21/6/R1/0000/21/7/R1/0000/21/8/R1/0000/22/1/M1/0000/22/2/E1/0000/22/3/E2/0000/22/4/L1/0000/22/5/R1/0000/22/6/R1/0000/22/7/R1/0000/22/8/R1/0000/23/1/R1/0000/23/2/R1/0000/23/3/R1/0000/23/4/R1/0000/23/5/M1/0000/23/6/E1/0000/23/7/E2/0000/23/8/L1/0000/24/1/R1/0000/24/2/R1/0000/24/3/R1/0000/24/4/R1/0000/24/5/M1/0000/24/6/E1/0000/24/7/E2/0000/24/8/L1/0000/&&&  8  Employee: 1 Total worked/rested: 16/8  Employee: 2 Total worked/rested: 16/8  Employee: 3 Total worked/rested: 16/8  Employee: 4 Total worked/rested: 16/8  Employee: 5 Total worked/rested: 8/16  Employee: 6 Total worked/rested: 8/16  Employee: 7 Total worked/rested: 8/16  Employee: 8 Total worked/rested: 8/16 |
| 8.4a | File size check |  |
| 8.4b | File size check |  |
| 8.5a | File extension check |  |
| 8.5b | File extension check |  |

# Roster Utility v0.1 User Guide

This user guide has been created to help staff members at Knatt Consulting Ltd to use the system for roster creation, validation and viewing.

|  |  |
| --- | --- |
| **Contents** | **Page no.** |
| Introduction to system | 2 |
| Minimum technical specifications | 2 |
| Installation procedure | 3 |
| Basic roster creation | 7 |
| Advanced roster creation – custom shifts | 11 |
| Viewing rosters | 13 |
| Clearing rosters | 15 |
| View help | 16 |
| View Knatt Consulting website | 16 |
| Quit the Utility | 17 |
| Troubleshooting | 18 |
| Uninstallation | 19 |
| Backup procedures | 20 |
| Legal | 20 |

## Welcome to Roster Utility!

### Introduction to system

The Roster Utility consists of a roster creation program and viewing utility. It can produce shift patterns for up to 8 employees for rosters which are up to 90 days in length.

In roster creation, you need to input the length of the roster, the number of employees, a custom shift pattern if appropriate, and other details concerning the penalty values for the rules. The program then finds an ideal solution for the problem. You simply choose a location to save the roster files, and at a later date different rosters can be compared.

You can also view rosters within the program: it has a ‘viewing utility’ which is built in.

Select this option, and you can load any roster created with the program. The roster is loaded into a large, clear grid in the centre of the program.

Being able to load different rosters gives you the capability to compare their advantages and disadvantages, and select the most appropriate one for your workplace needs.

### Minimum technical specifications:

* 1 gigahertz (GHz) or faster x86- or x64-bit processor
* Windows XP Service Pack 3 or later in either 32 or 64-bit
* 1 gigabyte (GB) RAM (32-bit); 2 gigabytes (GB) RAM (64-bit)
* 20MB hard drive space for installation of the program
* USB port available for transfer
* 1024x576 screen size
* Microsoft Office 2003 or later (or similar) installed to view documentation
* Internet Explorer 6; Mozilla Firefox 10.x; or Google Chrome 17.x for viewing web documents
* Internet connection recommended for backup to a cloud server

Please note that even if your system fits these requirements, the program is not guaranteed to run: some issues may arise with your configuration since every computer is set up differently. In this case, see the Troubleshooting section of this guide or contact your hardware manufacturer.

Roster Utility requires Microsoft’s .NET framework to be installed – this should be pre-installed on most systems. If you haven’t got it, you can download it for free from Microsoft at <http://www.microsoft.com/en-gb/download/details.aspx?id=17718>

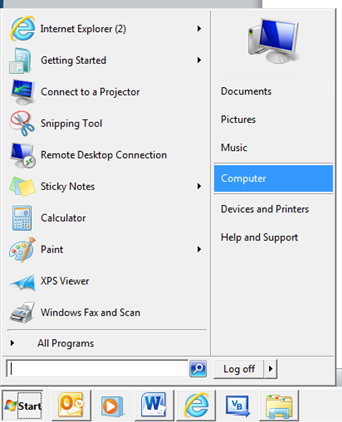
## Installation procedure

The Roster Utility is simple to install from the USB drive it is supplied on.

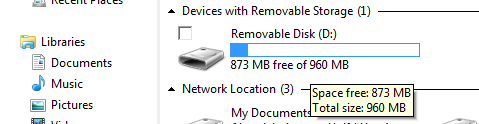
1. Locate one of your computer’s USB 2 or USB 3 ports. These may be on the front or back of the machine, or on the sides if you are using a laptop. Put the drive into one of these ports.
2. If the USB drive hasn’t been used on this computer before, a message may appear about installing drivers for the hardware. Select ‘OK’ or ‘Continue’ if prompted to install any drivers.
3. Open the Start Menu by clicking on the icon or by pressing the Windows key () located in the bottom left hand side of the keyboard.

 or 

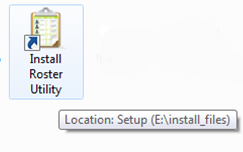
1. Click on ‘My Computer’ (Windows XP users) or ‘Computer’ (Windows Vista/7 users).

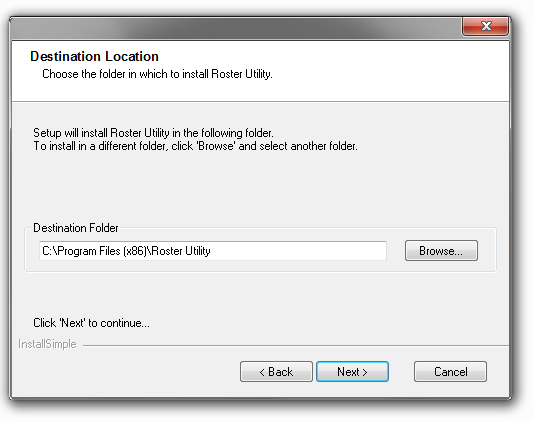
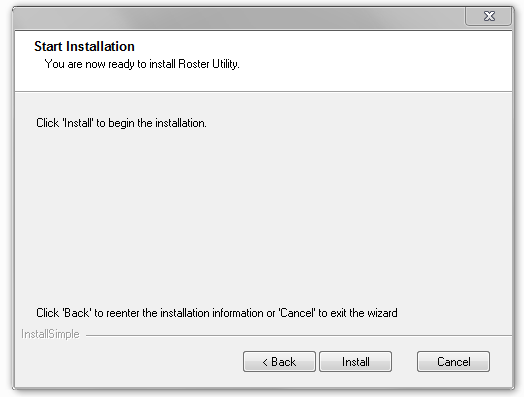


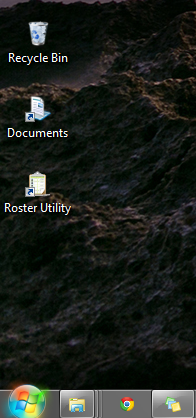
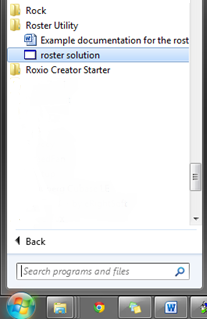
1. Double-click the icon for the USB drive. This will open the folder containing the program.



1. Double-click the shortcut file to run it.



1. The installer should open. Choose ‘Next’. 
2. Choose ‘Next’, after checking that the save location for the program files meets your requirements. The default save location will be fine for most users. 
3. Select ‘Install’ to install the program. 
4. The program will be installed, which may take several moments depending on the speed of your hardware.
5. When the program has finished installing, you will be presented with this screen. You can now exit the installer. 
6. After the program has installed, you can launch it by double-clicking on the icon on the Start Menu or on your desktop.

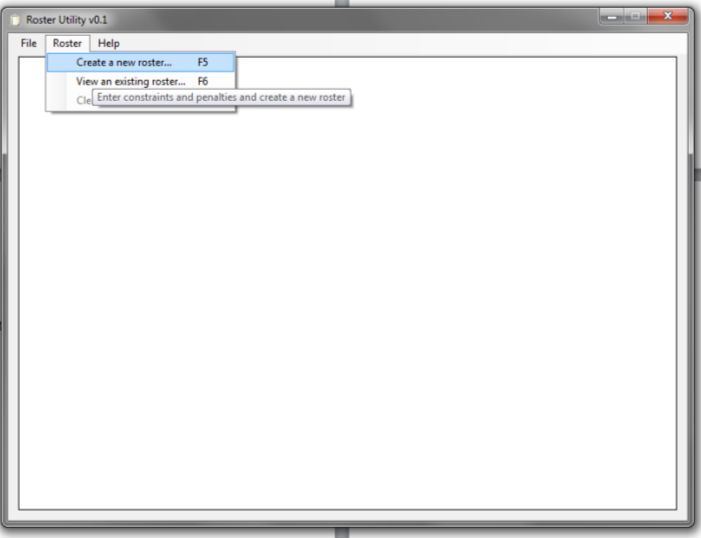
 

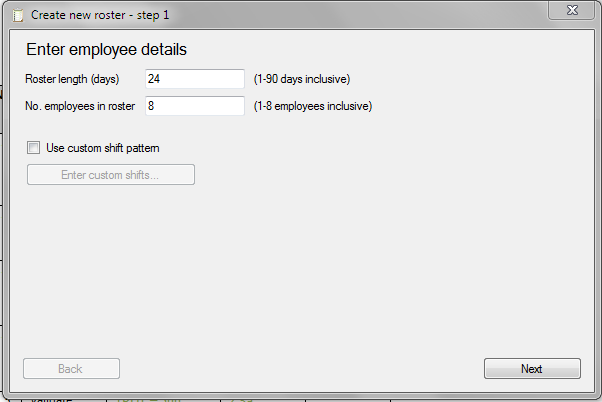
## Tasks within the system

### Creating and viewing a new roster

Creating new rosters is one of the major functions of Roster Utility, as the name suggests. Let’s create a basic roster first.

1. To start creating a new roster, choose *Roster > Create a new roster…* or press the F5 key on your keyboard.



1. On the first screen that appears, enter the number of days that your roster is going to run for (1-90 days inclusive) and the number of employees in the roster (1-8 inclusive). 



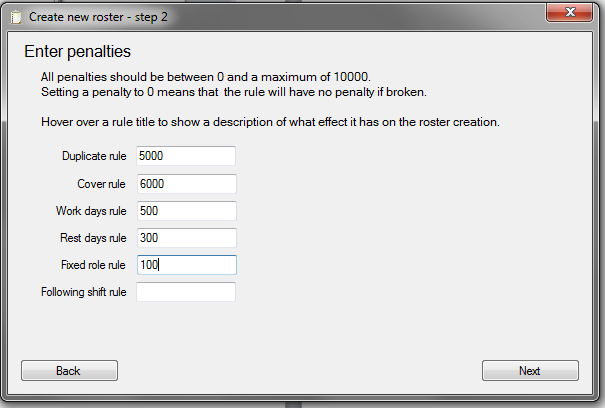
1. When you have done this, choose ‘Next’ to continue.



1. The next stage of creating a new roster is to enter penalties for each rule. For descriptions of what each rule does, please refer to the list.

The penalty will be incurred when the rule is broken by an allocation. Higher penalties will have a greater importance over lower ones. Setting a penalty to zero will mean that the rule is effectively not used.

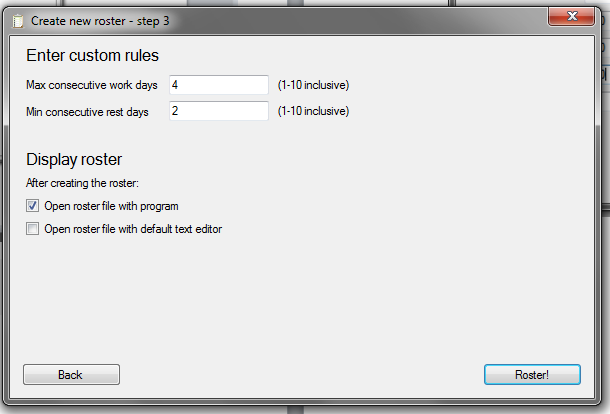
|  |  |
| --- | --- |
| Duplicate | If the program is not allocating a rest shift, and the number of previously allocated shifts is not zero, then the penalty is increased |
| Cover | If the program is going to allocate a rest shift but there are not enough employees to cover all of the work shifts if this happens, then the penalty is increased. |
| Work days | If the number of consecutive days worked is greater or equal to the maximum number of consecutive work days, and a work shift is assigned, the penalty is increased. |
| Rest days | If the number of consecutive days rested is greater or equal to the minimum number of consecutive rest days, and a work shift is assigned, the penalty is increased. |
| Fixed role | If the user has not entered custom shifts, employee 1 and 5 are only allowed to work shifts M and R. If they are assigned to another shift, the penalty is increased.  If the user has entered custom shifts, the program checks to see if it is assigning shifts that are not permitted by the user. If the program has to assign a shift that is not permitted by the user the penalty is increased. |
| Following shift | If the employee has previously worked either an M or L shift and is going to be assigned to an E1 or an E2, the penalty is increased. |



1. Select ‘Next’ when you are ready to continue.

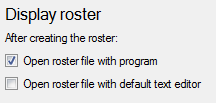


1. Now, enter the maximum number of consecutive work days (max no. of days that an employee can work in a row) and minimum number of consecutive rest days (minimum number of days that an employee must rest in a row). These both need to have values between 1-10 inclusive.



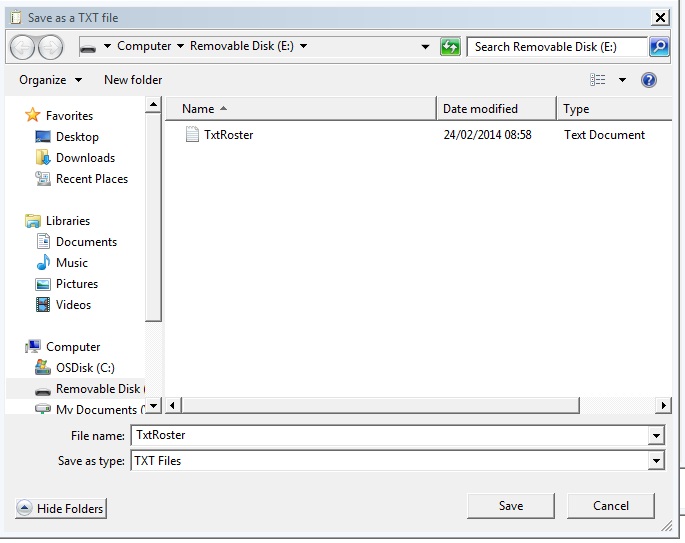
1. Now, select the display options. To have the roster file open within the special interface within the program (recommended for most rosters) you can check the first checkbox. If you wish to modify the roster file, you can check the second checkbox to have the roster file opened within your default text editor.

If you do not wish to view the roster at this time, you can leave both checkboxes unchecked – you will still be able to view the created roster at a later date.

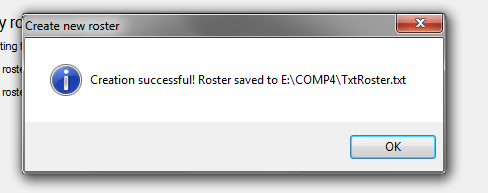


1. Click ‘Roster!’ to create the roster.
2. The program will now prompt you for a location in which to save the roster. This can be onto your hard drive, or into a folder that is synced with an online backup system (eg DropBox) for ease of file recovery should your computer hard drive fail. If a file already exists in this location, you will be asked if it should be overwritten.

Enter a file name and choose ‘Save’.



1. The roster will now be created and saved into the location you specified. When this is complete, you should see the following message (with your chosen file save location instead of the one shown):

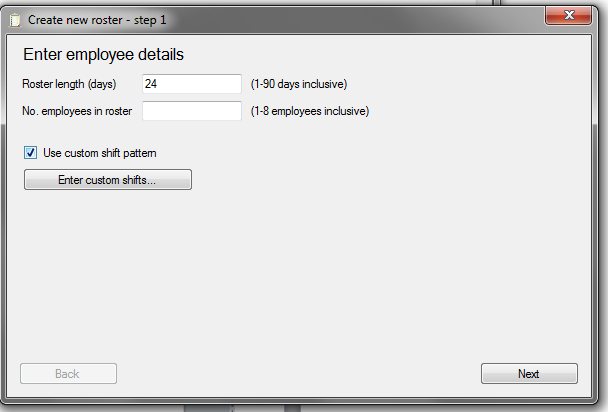


1. If you had selected to have the roster open in the program, or within a text editor, then the roster will be loaded into the program or editor respectively. See the section on loading rosters for assistance with this.

### More complicated rosters – custom shift patterns

Roster Utility is a tool which can also be used to enter custom shift patterns for the ‘fixed role rule’. It allows you to specify which employees can work which shifts instead of loading the default shift pattern.

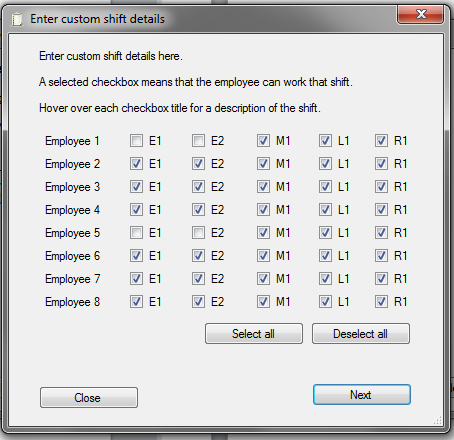
1. To enter a custom shift pattern, first create a new roster as detailed in the earlier section on creating rosters. When the first screen loads, check the checkbox labelled ‘Use custom shift pattern’. This will enable the ‘Enter custom shifts…’ button.



1. Click on the ‘Enter custom shifts…’ button.
2. You will be presented with the dialog box shown, which is where you can select which employees can work which shifts. A selected checkbox means that the employee can work that shift.

**Tip:** The ‘Select all’ and ‘Deselect all’ buttons may be useful at this stage.

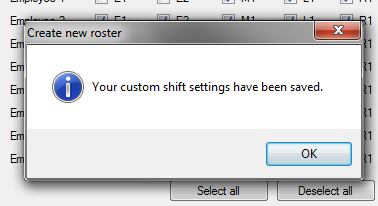
You can also hover over the checkbox titles (E1, E2…) for a description of the shift and an example of when the shift might be used within a roster.



1. Choose ‘Next’ when you have entered your custom shift pattern.



1. You will see the following message indicating that your shift pattern has been saved.

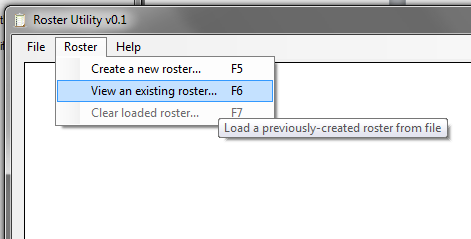


1. Now, you can continue the roster creation using the other screens in the roster creation part of the program. This is detailed in the previous section of the guide, ‘Creating and viewing a new roster’.

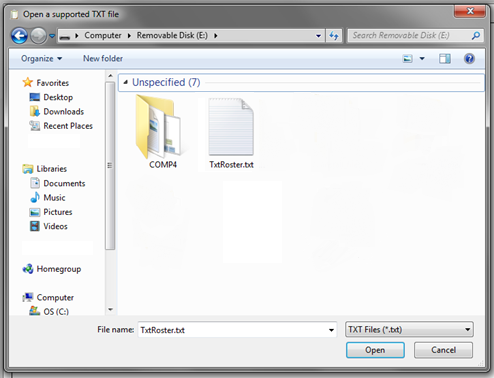
### Viewing rosters

Roster Utility also allows you to open and view rosters within its interface. This guide assumes you have previously created a roster using the steps detailed in the earlier part of the guide. If not, return to them and create a roster before attempting to view it.

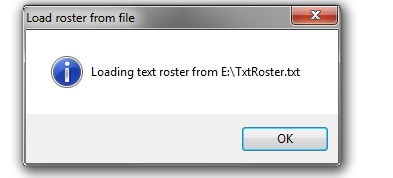
1. To open a previously-created roster, choose *Roster > View an existing roster…* or press F6 on the keyboard.



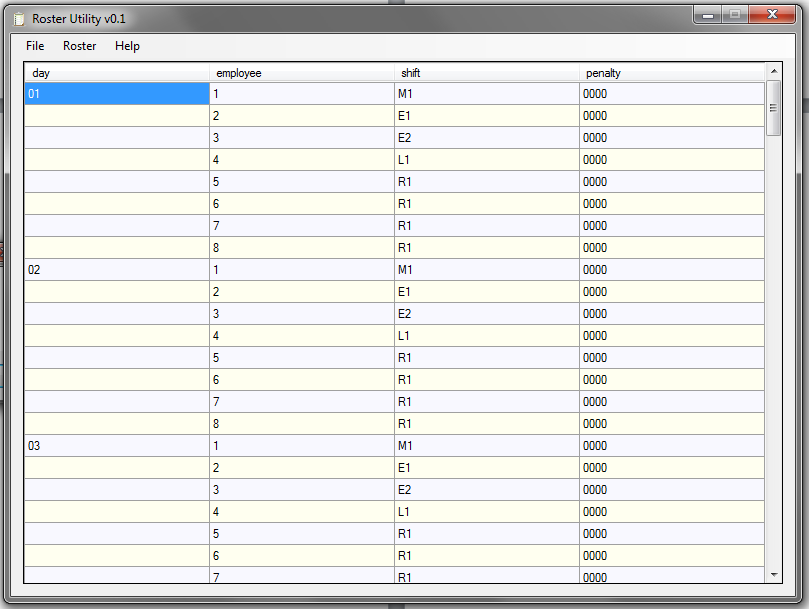
1. You will be prompted to open a roster file from its save location. Please only load roster files created using this software. Double click on a file or enter its file name to select it.



1. You will be presented with a message confirming the file is being loaded.



1. You will now see the roster loaded into the program. The rosters are organized by day, employee and then shift, and they have a column showing the penalty incurred for each shift allocation (in an optimal solution, the penalty incurred for all of the choices will be zero, such as the solution shown below).



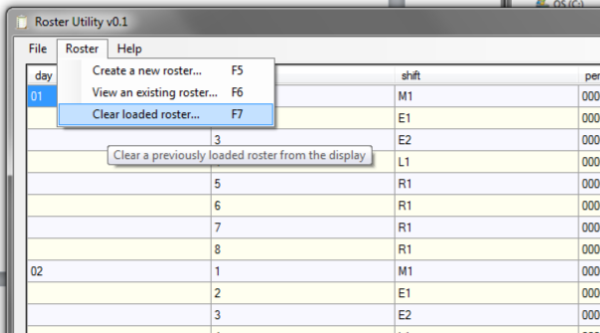
1. To clear the roster from the grid, refer to the following section detailing how to clear rosters.

**Tip:** You can load or clear rosters by right-clicking anywhere on the roster grid and selecting the relevant option from the context menu that appears.

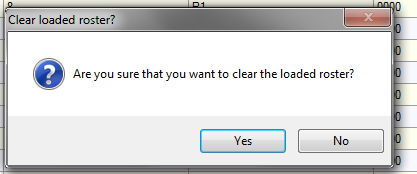
### Clearing rosters

Once a roster has been loaded into the grid, and you have finished with it, you may wish to clear it. Roster Utility provides an option for clearing the roster that does not require you to close the program and start again.

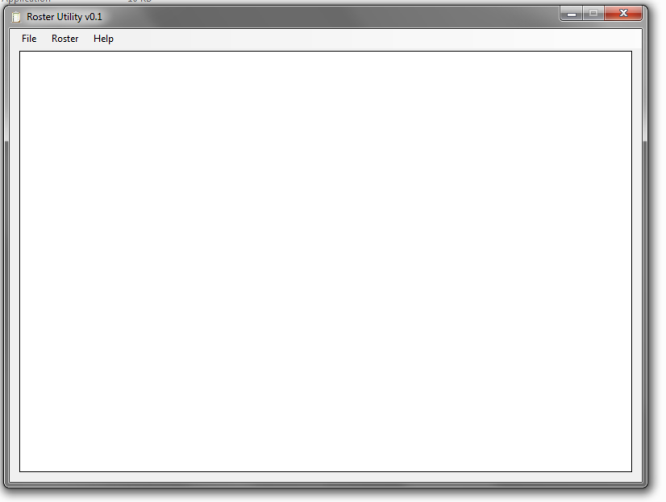
1. To clear a loaded roster, select *Roster > Clear loaded roster…*  or press F7 on the keyboard.



1. You need to confirm that you want to clear the loaded roster. Select ‘Yes’ if you are sure; otherwise, select ‘No’.



1. You should now see the cleared grid. You will not be able to clear a roster if there is not one currently loaded.

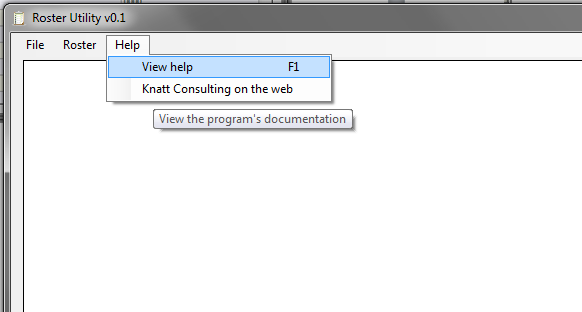


### Viewing help

This documentation can be accessed in digital form within the program itself, making it possible to reprint certain sections if they are lost or damaged. You may also find it easier to reference in a digital format.

You should have Microsoft Word or a program capable of reading .docx files installed before you attempt this, as suggested in the system requirements.

1. To view help, choose *Help > View help* from the menu at the top of the program, or press the F1 key on the keyboard.



1. This documentation file will load in your default editor for .docx files.

### View the Knatt Consulting website

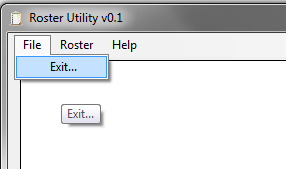
1. To view the Knatt Consulting website (<http://www.knattconsulting.com/>) from the program, choose *Help > Knatt Consulting on the web*.



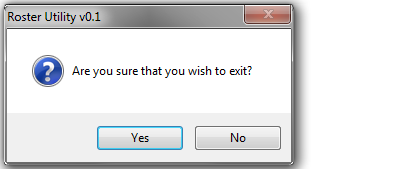
1. The default web browser will load and the web-page will be displayed.

### To quit the Utility

1. Once you have finished using the utility, it can be exited using the *File > Quit…*  command, by clicking the red X symbol in the top right corner, or by pressing the key combination ALT+F4.



1. You will need to confirm that you wish to exit: select ‘Yes’ if you are sure.



1. The program will now exit.

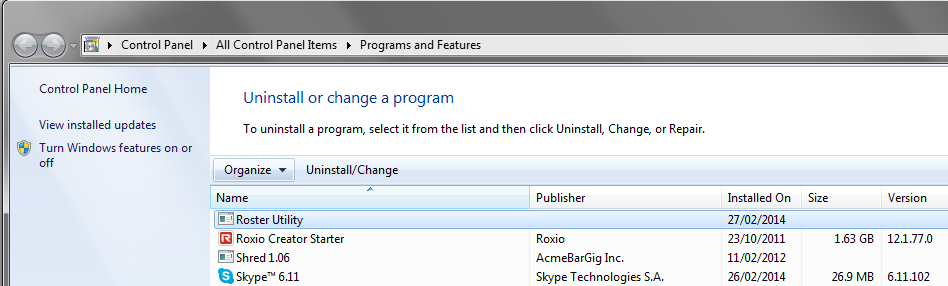
### Troubleshooting

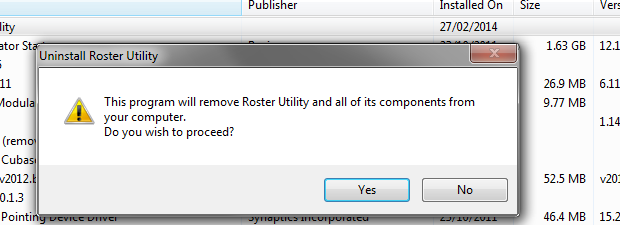
|  |  |
| --- | --- |
| **Error message** | **Explanation** |
|  | You have entered invalid data for the number of days or the number of employees. You need to change the values so they are between the values specified. |
|  | You have entered a penalty that is larger than the acceptable penalty value, resulting in an overflow. Please re-enter the penalty value according to the prompts displayed. |
|  | You have entered a value for max. consecutive work or min. consecutive rest that is too large or too small compared to the acceptable ones. |
|  | A file with the file name that you entered already exists. Please confirm that you want to replace the file by selecting ‘Yes’; otherwise, select ‘No’ and enter a different file name. |
|  | You have attempted to open the help file for the Utility but the program cannot find the file. This means that help cannot be loaded. If this problem persists, you may need to reinstall the program. |

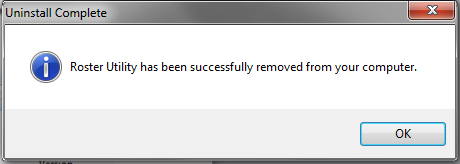
### How to uninstall the program

To remove Roster Utility from your computer, follow these steps:

1. Open the Start Menu, click on *Control Panel* and then *Add or Remove Programs.* You may also be able to access this command directly from the Start Menu in some versions of Windows.



1. Scroll down to Roster Utility in the list and select ‘Uninstall/Change’ from the bar above it, or from the right-click menu.
2. Confirm that you want to uninstall the program by selecting ‘Yes’. 
3. Once the program is uninstalled, you should see the following message:



## Backup Procedures

It is very important to back up files that you have created using the Roster Utility, as covered in our ‘introduction to the system’ meeting. This could be done manually (by copying and pasting them to a USB drive or external hard drive) or you could consider automated solutions, including Windows Backup.

Whichever method you use, it should be kept secure using password protection. Most cloud-based storage solutions incorporate this, and it can be found in the options for some memory sticks by right-clicking on the drive in My Computer, and selecting *Properties > Security.* If not, it can be set up using various free programs.

Cloud-based storage and backup solutions are detailed below:

* Google Docs/Google Drive:

<http://drive.google.com>

* Help for Google Docs/Google Drive:

<https://support.google.com/drive/?hl=en#topic=14940>

* DropBox:

<https://www.dropbox.com>

* Help for DropBox:

<https://www.dropbox.com/gs>

## Legal

1) GRANT OF LICENSE

This EULA grants you, the computer software end-user, the following rights:

1. The Utility may be used by anyone
2. You may install and use the Utility on any PC free of charge.
3. You may freely distribute the Utility but credit must be given to the original file source, and to the author of the program.
4. You may edit the source code of the program in any way for personal use only.

2) DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS

1. You may not remove program components from the Utility with intent to distribute them.
2. The Utility may not be distributed as a component of another commercial product.

3) COPYRIGHT AND TRADEMARKS

1. You must treat the Utility like any other copyrighted material for archival purposes only. You may not copy the Utility’s documentation except for personal use where appropriate.
2. You may not remove, modify or alter any part of the license.

4) MISCELLANEOUS

If you have any questions concerning this license, please contact the author through the contact details previously provided.

# Bibliography

Sources of third-party material used in the development of this project are listed below.

Kevin Bond and Sylvia Langfield (2009)

AQA A2 Computing, Nelson Thornes

Susie G Jameson (2008)

Edexcel Modular Mathematics: Decision 1, Pearson/Edexcel

Bastian Rieck

Basic Analysis of Bin-Packing Heuristics

<http://bastian.rieck.ru/uni/bin_packing/bin_packing.pdf>

York College of Pennsylvania

Lecture 8: Quicksort

<http://faculty.ycp.edu/~dbabcock/cs360/lectures/lecture08.html>

Eindhoven University

Lecture 13: The Knapsack Problem

<http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf>

David Pisinger

Quadknap: A callable routine for solving quadratic knapsack problems in C

<http://www.diku.dk/~pisinger/quadknap.c>

Vincent Cho, Gene Pak Kit Wu and W.H. Ip (2009).

An Aircraft Service Staff Rostering using a Hybrid GRASP Algorithm, International Journal of Engineering Business Management , Wai Hung Ip (Ed.) <http://www.intechopen.com/journals/international_journal_of_engineering_business_management/an_aircraft_service_staff_rostering_using_a_hybrid_grasp_algorithm>