**Scheduled Harmony Instruction Manual**

Thank you for using Scheduled Harmony! Scheduled Harmony is an application designed for Windows that creates period-based schedules for both students and teachers. Scheduled Harmony considers student course requests and teachers’ courses to create ideal schedules customized to the school. This document is a comprehensive guide to the use of Scheduled Harmony, referred to as ‘the program.’

**How The Program Works**

The program uses both ‘input files’ and ‘output files.’ Input files are created by the user using relevant real-world data. The program will never modify these files. Output files display the results of running the program. The program may create and/or modify these files as necessary. ‘Data files’ refers to both types of files.

The program contains two ‘executables.’ These are application files with the “.exe” file type. The executables will only run on a Windows computer. Executables can be run by double-clicking on the file to open it. A window will pop up and the program will begin to run.

As the program starts, it will attempt to load the data from the input files into the program’s memory. If any data in any file is formatted incorrectly or if the data the file contains is incompatible with the program in any way, the program may discard that data or an error may occur. If data gets discarded, an output file should contain information about what was discarded and why. If any output files already exist, the program will then attempt to load the saved schedule configuration.

After the data has been successfully loaded into the program’s memory, the program will attempt to create a ‘schedule.’ A schedule is an assignment of students and teachers to courses and periods such that each student and each teacher has a course assigned to them for every period. These courses may include various types of ‘free periods’ where the individual isn’t obligated to learn or teach. A ‘class’ is the specific occurrence of a course during a specific period. Ideally it contains exactly one teacher and a number of students that is within the range provided to the algorithm.

Each schedule is assigned a ‘cost’ by the program. A schedule’s cost attempts to quantify how close the students are to having classes of their requested courses. A lower cost means that more students got the classes they requested. Each unit of cost roughly corresponds to a class of students not getting one of their requests for a single semester. One schedule is considered better than another schedule if its cost is lower.

The user can set how long the program should run for in the program settings file. Running the program for a longer amount of time will make it more likely that better schedules are found. Once the program is finished running, it will determine whether the best schedule found is better than the schedule that was previously saved in the output files. If the program found a better schedule, it will write the found schedule to the output files, overwriting any previous information contained in these files.

Any errors that occur will cause the program to stop running but will not affect any saved data. Error messages may be viewed by running the executable “SchedulingTest.exe”. Many of the possible error messages will clearly inform the user of what the problem is and/or what is causing the problem. However, some error messages may be difficult to interpret. For questions about the program or specific error messages, the creator of the program may be contacted using the contact information located at the bottom of this document.

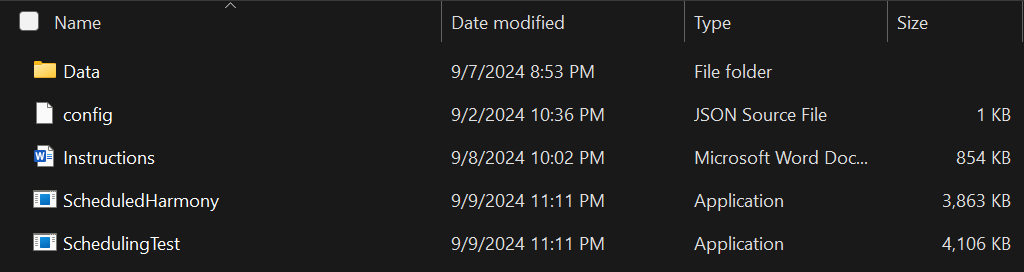
The program only works with exactly two semesters, each with exactly eight periods. To request a version of the program that uses a different number of periods, the creator of the program may be contacted.

**Folder Structure**

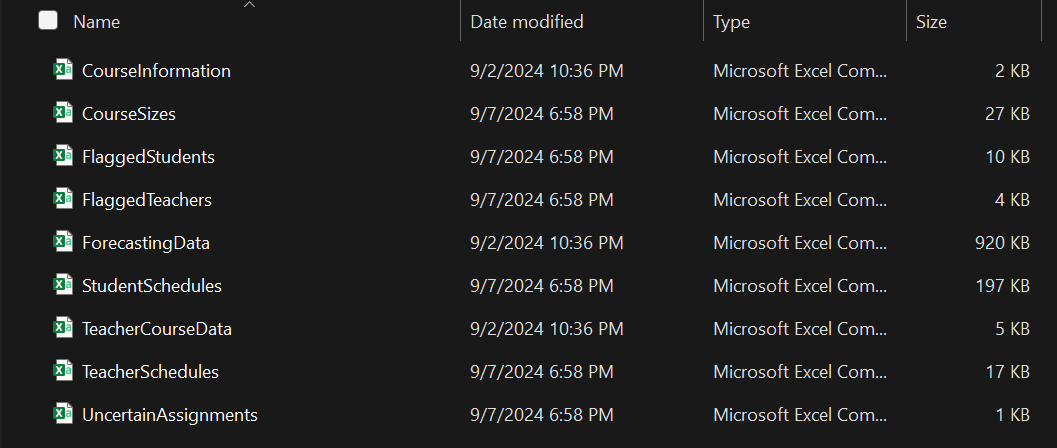
All components of the program should be contained within a single folder and/or its subfolders. This single folder is the ‘parent folder.’ The parent folder may be named anything. If a folder or file is said to be located in the parent folder, it must be directly within the parent folder rather than in any of its subfolders. Files not mentioned in this document may be placed in the parent folder or its subfolders without disrupting the program. Any such files won’t be used or modified by the program.

All data files should be placed in a folder named exactly “Data.” The folder named “Data” should be placed in the parent folder. Each data file must have the exact name described in this document. Each data file must be saved in CSV format. A CSV file is a plain-text version of a spreadsheet. To save an Excel spreadsheet as a CSV, go to File → Export → Change File Type → CSV (Comma delimited). To save a Google Sheet as a CSV file, go to File → Download → Comma Separated Values (.csv). All folder and file names are case-sensitive.

The file named “config.json” contains program settings that may be changed by the user to affect how the program runs. This file should be placed in the parent folder.



Parent Folder Structure



Data Folder Structure

**The Executables**

The application files that the user opens to run the program are the ‘executables.’ To open an executable, the user should double-click on the executable file. Any time the user opens an executable file, they are considered to be running the program. The parent folder should contain two executables, named “ScheduledHarmony.exe” and “SchedulingTest.exe”. Application windows may be closed at any time without an effect on previously saved data, but their results won’t be saved unless the executable has finished running. If the computer falls asleep while an executable is running, the program will be either paused or stopped. To prevent this from happening, the computer’s sleep settings should be changed so that the computer doesn’t fall asleep during the time that an executable is running. Once an executable has finished running, the user may press enter or close the application window to safely exit the program.

ScheduledHarmony is the executable that should be used to find optimal schedules. Any errors that occur while running this executable will cause the program to terminate and the application window to close. This executable will run for a set amount of time without saving any results or modifying any data files until the end of the time period. The amount of time it runs for can be set in the file “config.json”. While running, a new line of text will be printed in the window every minute.



The number of iterations is cumulative and informs the user how efficiently the program is running. The program will run more efficiently and have a higher iteration count if the application window is open and in focus. A typical number of iterations per minute is between 500 and 3000 depending on computer conditions.

SchedulingTest is a program that validates the user’s input data and ensures that the algorithm will be able to use it without any issues. It is recommended that the user runs this executable first after making any changes to the input files. Any errors that occur during these checks will be displayed in the application window. A successful run may look something like the following.

A screen shot of a computer

Description automatically generated

Running SchedulingTest Without Errors

The line “Student requests granted” tells the user how many optimized student requests resulted in a class size within the given range, out of the total number of optimized requests. Optimized requests shouldn’t include free periods.

**Input Files**

Input files are created by the user and the program will never modify these files. The program uses a total of three input files, named “CourseInformation.csv”, “ForecastingData.csv” and “TeacherCourseData.csv”. These names are case-sensitive, and all three of these files must exist for the program to run successfully.

Input files may never contain the comma character “,” for any reason other than to separate columns. If the user is creating or editing an input file in a spreadsheet application, they should never put the comma character in any cell. The inclusion of the comma character within any field of any input file will prevent the program from parsing the file correctly.

**CourseInformation File**

This file contains information about specific courses where one or more default values shouldn’t apply. Any course codes not present in this file will be linked automatically based on the code and name present in ForecastingData, and its information will be set to default values. This file may be empty.

Through this file, the user may customize how course codes get linked, remove redundant codes, specify a unique minimum and maximum student capacity, and limit which periods the course is offered. Any field may be left blank unless marked by an asterisk below. Any non-required field left blank either won’t be used or will be assigned a default value.

If the program is unable to parse any row, an error will occur. Any changes to this file will likely change the cost of a saved schedule, which increases the chance that the saved schedule gets overwritten while running either executable.

The character “#” as the first character of the first cell in a row will cause that row to be ignored. This file must contain a header row.



Header Row of CourseInformation.csv

**CRN\*:** The code corresponding to the course that should be used in the output files.

**CRN 2:** Any other code corresponding to the same course as CRN that should be replaced by CRN in the output files. If the field Name contains any text, only requests with a name that matches Name will be replaced.

**Name:** The name that should correspond to the course. If Name doesn’t match the name found in ForecastingData, the course’s name will be overwritten to Name. If CRN 2 contains a course code, only requests with Name will have their course code replaced.

**Linked CRN\*\*:** If CRN corresponds to only one semester of a year-long course, the CRN of the other semester must be placed here to ensure that the two course codes get linked.

**Linked CRN 2:** Any other code corresponding to the same semester as Linked CRN that should get replaced by Linked CRN in the output files. If the field Linked Name contains text, only requests with a name that matches Linked Name will be replaced.

**Linked Name:** The name that should correspond to the same semester as Linked CRN. If Linked Name doesn’t match the name found in ForecastingData, the course’s name will be overwritten to Linked Name. If Linked CRN 2 contains a course code, only requests with Name will have their course code replaced.

**Semester\*:** A number code that designates which semesters the course should be offered and how it should be linked. This field is case insensitive.

**0** or **DELETE** - CRN will be removed from each student’s requests.

**1** or **FIRST** - CRN corresponds to the first semester of a year-long course.

**2** or **SECOND** - CRN corresponds to the second semester of a year-long course.

**3** or **SEMESTER** - CRN corresponds to a half-credit semester-long course.

**4** or **YEAR** - CRN corresponds to a course that may be taken as either a semester-long or year-long course. If a student has two of CRN in their requests, the first and second semester of the course will be during the same period. CRN will be modified to end in “-\*”, “-1” or “-2”.

Any other text in this field will be treated as “4”. Codes “1” and “2” may also be used to restrict a semester-long course to only be offered one semester. If this method is used, a unique Linked CRN and Linked Name must be entered into their respective fields.

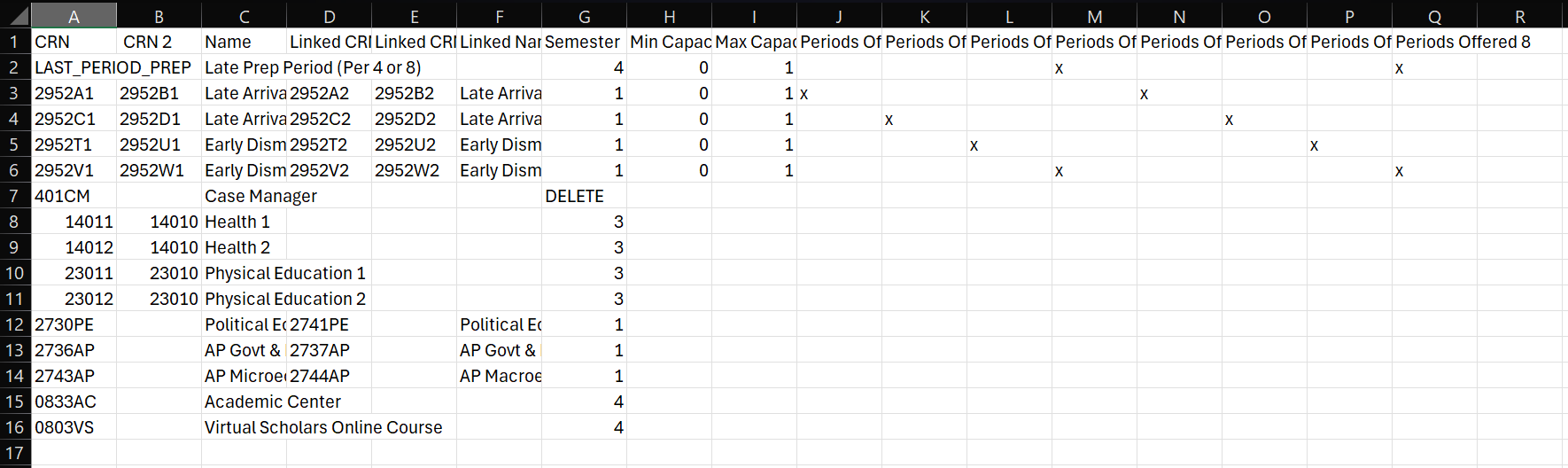
**Min Capacity:** Overrides the global minimum class size to optimize for. Must be a positive integer less than or equal to Max Capacity.

**Max Capacity:** Overrides the global maximum class size to optimize for. Must be a positive integer greater than or equal to Min Capacity. If Min Capacity is exactly “0” and Max Capacity is exactly “1”, the program won’t attempt to create classes for the course. This is recommended for all types of free period.

**Periods Offered 1-8:** Any text other than exactly “0” in any of these cells will limit the periods the course can be scheduled for to only the periods corresponding to the cells with marks. If all of these cells are blank, the course may be scheduled for any period.

\*Required

\*\*Required for courses with distinct first and second semester codes



CourseInformation.csv

**ForecastingData File**

This file contains the course requests of each student made during forecasting at that school. This data can most likely be acquired from StudentVue [process currently unknown]. Each line of this file defines a single course request from a single student. Each student should have multiple lines to define each of their course requests. These requests should be for both first and second semester courses.

The character “#” as the first character of the first cell in a row will cause that row to be ignored. This file cannot contain a header row.

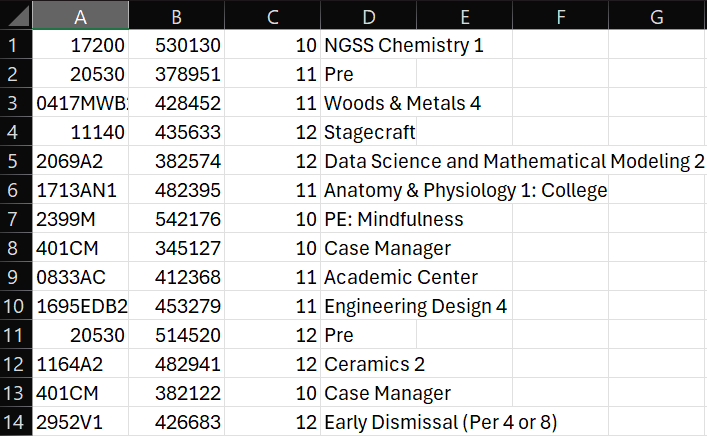
**Column 1:** The code corresponding to the requested course, referred to as ‘course code’ or ‘CRN.’

**Column 2:** Student ID the request is associated with. Must be a positive 6-digit integer.

**Column 3:** Student grade level. This column is currently unused and may be blank. This column must exist for the file to be parsed correctly.

**Column 4:** The name of the course corresponding to the CRN.

The CRNs and names present in this file will be used by the program to load courses and their relevant information into the program’s memory. Pairs of CRNs with their sole difference being the final character either as “1” or “2” will be linked and treated as two semesters of a year-long course. Pairs of names will be similarly linked regardless of their course codes. The file CourseInformation can be used to override these automatic linkages.



ForecastingData.csv

**TeacherCourseData File**

This file contains the courses each teacher will be teaching, as well as the number of classes of each course they will be teaching. One ‘A day’ and one ‘B day’ prep period will be automatically added to each teacher for both semesters.

The character “#” as the first character of the first cell in a row will cause that row to be ignored. This file must contain a header row.



Header row of TeacherCourseData.csv

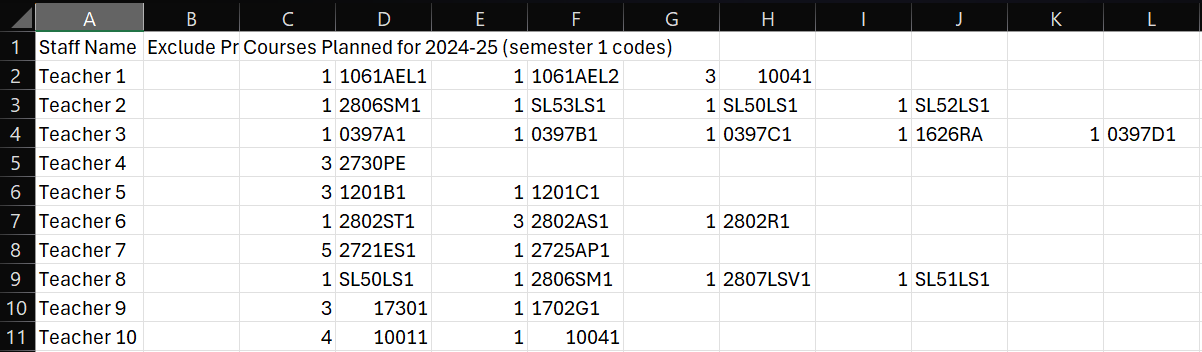
**Column 1:** Some unique identifying string corresponding to the teacher, referred to as ‘Teacher ID.’

**Column 2:** Any text in this field will prevent a teacher’s default prep periods from being automatically added to their courses.

**Columns 3+:** Text in these fields should alternate between a number and a course code. The number should be placed in the column directly before the course code to define how many classes of that course the teacher will be teaching. For any year-long courses with linked first and second semester codes, either code may be entered and an equal number of the linked code will be automatically added. Any other courses assigned to the teacher will be counted on a per-semester basis. For example, if the number “4” is placed before a course code corresponding to a semester-long course, the program may schedule the teacher to teach 2 classes of the course first semester and 2 classes of the course second semester.

To manually assign unique prep periods (for example 4th or 8th prep only) to a teacher, the unique prep period information should be entered into CourseInformation. The corresponding course code should be entered in the teacher’s row. A mark of any type should then be placed in the “Exclude Prep Periods (Override)” column of the teacher’s row. The unique prep period may be used for multiple teachers.

To manually assign a teacher to teach on only A days or only B days, a unique free period should be entered into CourseInformation with its periods limited to only the day that the teacher does not wish to teach. A mark of any type should be placed in the “Exclude Prep Periods (Override)” column of the teacher’s row. The teacher should then be assigned one copy of a prep period on the day that they are teaching and four copies of the unique free period. The unique free period may be used for multiple teachers.



TeacherCourseData.csv

**Output Files**

Output files display the results of running the program. If an output file doesn’t exist as the program attempts to save its results, the file will be created. If it does already exist, the file will be overwritten. If an output file already exists and is being used by another application on the computer, the program may prompt the user to close any such applications or an error may occur. If this executable is located in a protected folder on the computer, the user may have to right-click on the executable and select “Run as administrator” to give the executable permission to create and edit these files.

The program will create and/or overwrite a total of six output files when run successfully, named “UncertainAssignments.csv”, “FlaggedStudents.csv”, “FlaggedTeachers.csv”, “StudentSchedules.csv”, “TeacherSchedules.csv” and “CourseSizes.csv”. These files do not have to exist for the program to run successfully.

If the files “StudentSchedules.csv” and “TeacherSchedules.csv” do exist as the program starts, it will attempt to use their contents to load the previously saved schedule into the program’s memory. If a better schedule than the one found in these files is ever created by the program, it will overwrite the contents of these files. This will delete all information about the previously saved schedule.

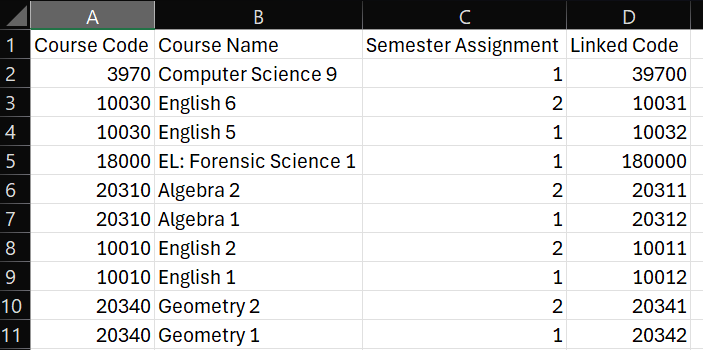
The user should save a copy of all data files to a different location on the computer if they wish to permanently store a schedule configuration.

**UncertainAssignments File**

As the program starts, it will use the input files to determine whether each course code should be linked to another course code to form a year-long course. Pairs of course codes with their sole difference being the final character either as “1” or “2” will be linked and treated as two semesters of a year-long course. Pairs of names will be similarly linked regardless of their course codes. All other course codes will be treated as corresponding to semester-long courses and will not be linked to a course code.

Any pairs of courses that were linked by name will be displayed in this file. For each course that was linked this way, the number of the semester it was assigned to and the code it was linked to will be displayed in columns 3 and 4 respectively. The semester assignment column (col. 3) may also contain the number “3”, which represents an unlinked semester-long course.

Any course codes found in ForecastingData with multiple associated names may be displayed in this file. Similarly, any names with multiple associated course codes found in ForecastingData may be displayed in this file. If all the information in this file is correct, the user may leave these courses unmodified.



UncertainAssignments.csv

**FlaggedStudents File**

As the program starts, it will use the contents of ForecastingData to create groups of requests assigned to each student. If the program determines that the student may have made a mistake during forecasting or requested a set of courses that is impossible to take, it will display the student in this file. The student may still be included in the schedule the program creates, but their classes may not match what they intended to forecast for. The user should manually check each student displayed in this file and correct any errors found in their forecasting requests.



Header row of FlaggedStudents.csv

**ID:** The student’s ID number.

**Included in Schedule:** This field either contains “1” or “0”. “1” means that the student was successfully included in the schedule and placed into classes, but their requests may still be incorrect. “0” means that the program was unable to include the student in the schedule, and that student was not placed in any classes.

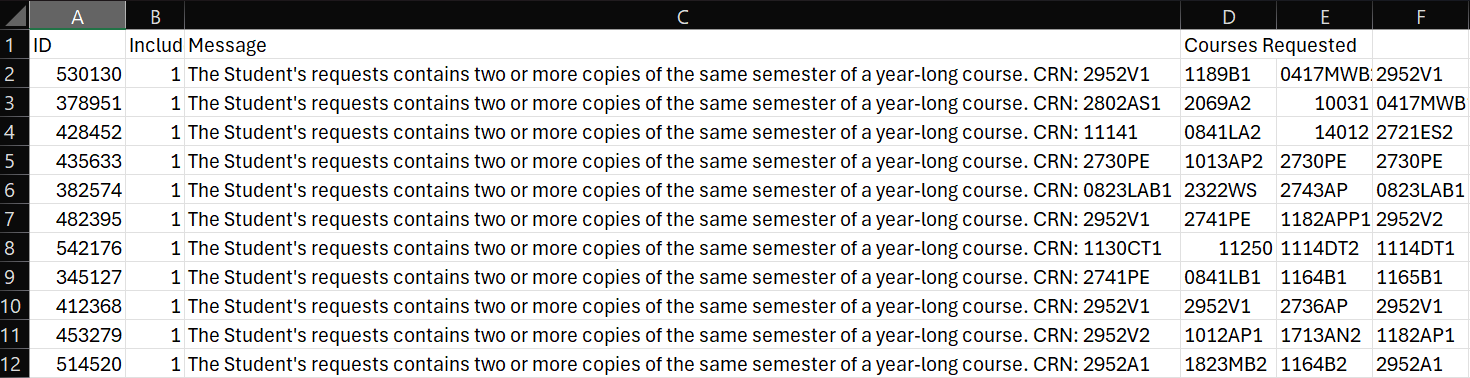
**Message:** A sentence that describes the issue with the student’s requests. There are three common types of messages.

**Two copies of half of a year-long course:** This most commonly occurs when a student erroneously requests two copies of the first semester of a year-long course, rather than one copy of the first semester and one copy of the second semester. The code with two copies should be listed in the message.

**Wrong number of courses:** This occurs when a student requests more course codes than there are periods. If the program is able to include the student in the schedule, duplicate course codes or the final course code of their requests may be removed. Any courses that get removed will not be listed in the message. If a student requests fewer course codes than there are periods each semester, additional blank periods will be added to their requests and the student will not be listed in this file.

**Unable to assign courses:** Some aspect of the student’s requests makes it so that the program is unable to put the student into a period for each of their requests. This could be because the student has requested more courses offered only during first semester than there are periods that semester. It may also be because the courses in CourseInformation restrict the periods they are offered to the point that students are unable to take some combination of courses. The message will include a guess of which request is causing the problem that may or may not be accurate.

**Courses Requested:** A list of all the course codes requested by the student that the program found in ForecastingData.



FlaggedStudents.csv

**FlaggedTeachers File**

As the program starts, it will use the contents of TeacherCourseData to assign courses to teachers. If the program detects any problems while assigning courses to a teacher, it will display the teacher in this file.



Header row of FlaggedTeachers.csv

**ID:** The identifying string corresponding to the teacher.

**Included in Schedule:** This field either contains “1” or “0”. “1” means that the teacher was successfully included in the schedule and assigned to teach classes, but their classes may be incorrect. “0” means that the program was unable to include the teacher in the schedule, and that teacher was not assigned to teach any classes.

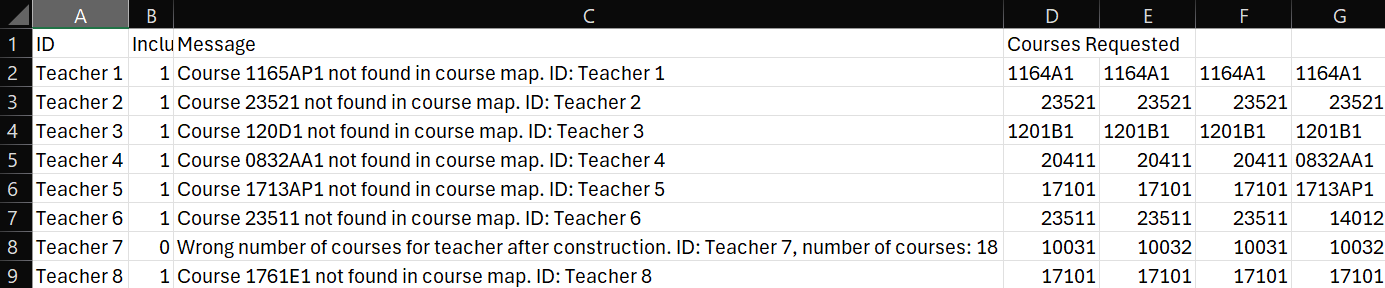
**Message:** A sentence that describes the issue with the teacher’s assigned courses. There are three common types of messages.

**Course not found in course map:** The program found no instances of a given course code within ForecastingData or CourseInformation. The course code will be listed in the message. The teacher will be included in the schedule, but will not have any instances of that course code in the classes they are assigned to.

**Wrong number of courses:** This occurs when a teacher is assigned to teach more classes than there are periods. The program will not include the teacher in the schedule. If a teacher is assigned fewer classes than there are periods each semester, additional blank periods will be assigned to them and the teacher will not be listed in this file.

**Unable to assign courses:** Some aspect of the teacher’s assigned courses makes it so that the program is unable to assign a period to each course. This could be because the teacher has been assigned more courses offered only during first semester than there are periods that semester. It may also be because the courses in CourseInformation restrict the periods they are offered to the point that the teacher is unable to be assigned all of their courses. The message will include a guess of which course is causing the problem that may or may not be accurate.

**Courses Requested:** A list of all the course codes assigned to the teacher that the program found in TeacherCourseData. This list of codes may or may not include linked codes for year-long courses, depending on the message type.



FlaggedTeachers.csv

**StudentSchedules File**

This file displays the classes each student is assigned to each period. Each row corresponds to a single student. The student’s ID is listed in the first column. Columns to the right of the first column display the student’s classes ordered by period. The first half of the class list contains classes the student is assigned to during first semester, and the second half of the class list contains classes the student is assigned to during second semester.

Each period the student is placed into may or may not have enough students and/or teachers to create a full class. It is the user’s responsibility to determine which courses will ultimately be offered each period. Students with fewer requests than there are periods each semester will be assigned the default course code “NO\_COURSE\_ASSIGNED” for some number of their periods. It is the user’s responsibility to fill these empty periods with class assignments.

Any modifications made to this file by the user may prevent a previously saved schedule from loading correctly as the program starts to run.

A screenshot of a computer

Description automatically generated

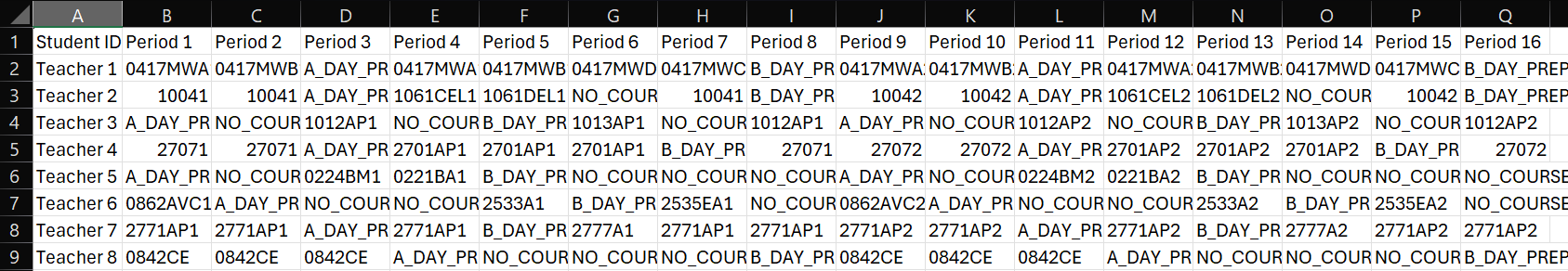
StudentSchedules.csv

**TeacherSchedules File**

This file displays the classes each teacher is assigned to teach each period. Each row corresponds to a single teacher. The teacher’s ID is listed in the first column. Columns to the right of the first column display the teacher’s classes ordered by period. The first half of the class list contains classes the teacher is assigned to during first semester, and the second half of the class list contains classes the teacher is assigned to during second semester.

Each period the teacher is assigned to may or may not have enough students to create a full class. It is the user’s responsibility to determine which courses will ultimately be offered each period. Teachers with fewer assigned classes than there are periods each semester will be assigned the default course code “NO\_COURSE\_ASSIGNED” for some number of their periods. It is the user’s responsibility to fill these empty periods with class assignments.

Any modifications made to this file by the user may prevent a previously saved schedule from loading correctly as the program starts to run.



TeacherSchedules.csv

**CourseSizes File**

This file displays the number of students and teachers assigned to each period a course could be offered. Each row corresponds to a single course code. Linked codes are displayed separately. Cells that contain a count of both students and teachers display the student count on the left, followed by “ | ”, followed by the teacher count on the right. Each cell in the second row of this file contains the sum of all of the numbers below it. These sums do include free periods.



Header row of CourseSizes.csv

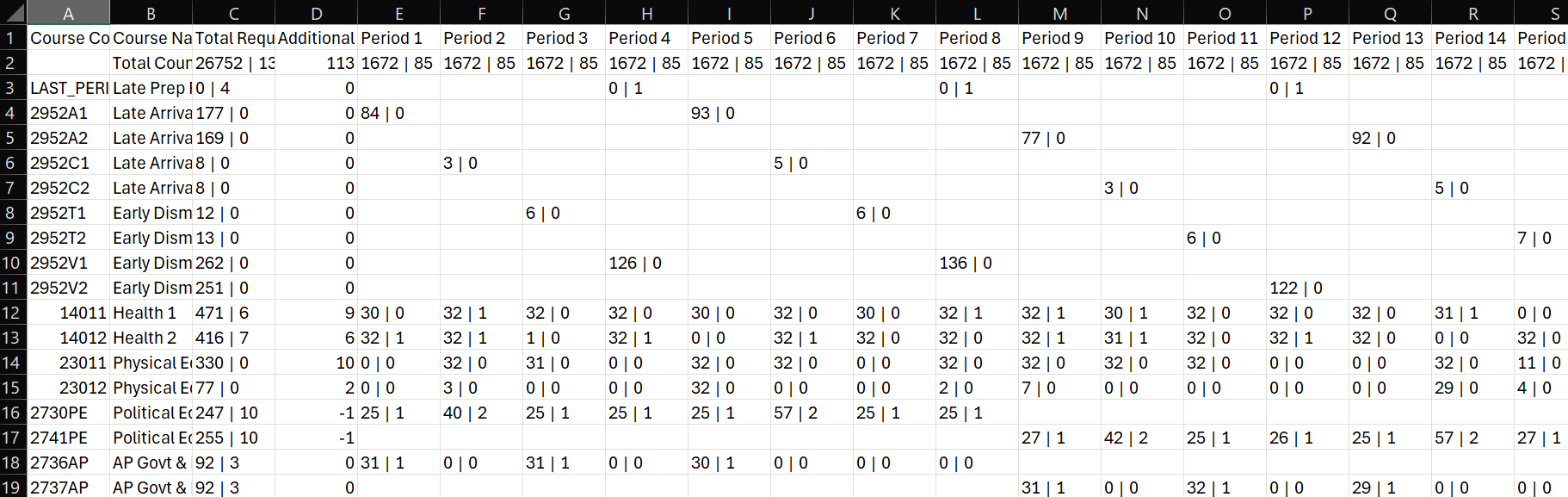
**Course Code:** The code that corresponds to the course.

**Course Name:** The name associated with the course code. May be blank if the program doesn’t find an associated name in the input files.

**Total Requests:** Contains the total number of students that requested the course, and the total number of teachers assigned to teach that course.

**Additional Teacher-Periods Needed:** The program’s estimation of the number additional classes that teachers should be assigned to. A positive number indicates that more teachers are needed for the course. A negative number indicates that there are too many teachers for the course and that many assignments should be removed from teachers. These numbers are recommendations only.

**Periods 1+:** Contains the number of students and the number of teachers that are assigned to the course during that period. If a period’s cell is blank, the course is not offered during that period.

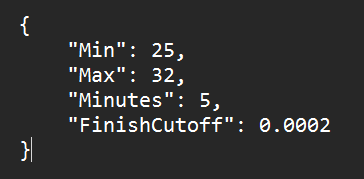


CourseSizes.csv

**Program Settings**

‘Program settings’ refers to the file named “configs.json”. This file should be located in the parent folder. This file’s contents should consist of an object (defined by curly braces) with the properties “Min”, “Max”, “Minutes” and “FinishCutoff”. The user may change these properties to affect how the program runs.

If the program settings file doesn’t exist as the program begins to run, it is generated automatically with default values. If one or more properties are missing, they will be automatically assigned the default value and added to the file. If an error occurs while reading the file, the user may delete the file to have it regenerated with default values. These default values are pictured below.



config.json

**Min:** The default minimum number of students per class. This number is assigned to each course not defined in CourseInformation, and is the default if the minimum field of a course in CourseInformation is blank. Min must be a positive integer.

**Max:** The default maximum number of students per class. This number is assigned to each course not defined in CourseInformation, and is the default if the maximum field of a course in CourseInformation is blank. Max must be a positive integer.

**Minutes:** The number of minutes the user wishes the executable ScheduledHarmony to run for. A larger number of minutes will increase the chance that the final schedule has a lower cost. Minutes must be a positive integer.

**FinishCutoff:** Different values of this number can change how effectively the program is able to create a low-cost schedule. The optimal value of this number depends on the number of students in the schedule and may depend on other unknown factors. For a school of 1500 to 2000 students, the optimal value is likely close to “0.0002”. For larger schools the optimal value will be lower, and for smaller schools the optimal value will be higher. The user is encouraged to test how the program performs with different values. FinishCutoff must be a number between 0 and 1 and contain a decimal.

**Errors**

If the data contained in the input files is incompatible in any way with the program, an error may occur. To view an error message, the user should run the executable SchedulingTest. Any errors that occur while the executable ScheduledHarmony is running should also occur while SchedulingTest is running. There are several different types of errors that may occur. All error messages should include both one of the error types listed below and a specific message.

**Data File Path Error:** Either an input file couldn’t be found, or an output file couldn’t be written to. If an input file couldn’t be found, the user should check to make sure that the specified file exists inside a folder named exactly “Data” located in the parent folder. The path to the parent folder is the second line in the application window that opens when the executable SchedulingTest is run. All file and folder names are case-sensitive. If an output file couldn’t be written to, the user should make sure that the file isn’t currently open in another application. They can also check if the parent folder is in a protected location within the computer. If the parent folder’s writing access is limited to administrators only, the user should right click on the executable and select “Run as administrator”.

**Course Information Error:** Some value in the file CourseInformation is incompatible with the program. Some additional information about the error should be displayed, including a line number and/or course code. The line number refers to the row of CourseInformation that contains the invalid information. The course code refers either to the value of the first cell in the row that contains the invalid information or a value in the “Linked CRN” field of a row.

**Request Information Error:** Some value in either the file ForecastingData or the file TeacherCourseData is incompatible with the program. The error should specify which file the invalid information is located within, as well as provide some additional information about the error. If the error refers to a “Student”, the invalid information is located within ForecastingData. If the error refers to a “Teacher”, the invalid information is located within TeacherCourseData. The error should also contain a line number and/or ID. The line number refers to the row of the file that contains the invalid information. The ID refers to the value of the first cell in the row that contains the invalid information.

**Program Settings JSON Error:** The format of the program settings file (named “configs.json”) or a value of one of its properties is invalid. The user can delete the file to have it regenerated with default values the next time the program is run.

**Unknown Error:** Some internal error has occurred within the program. The error message will most likely be undescriptive or fail to contain the location of the problem. The user should run the program again to determine if the error is repeatable. If the issue persists, it is recommended that the user seeks assistance using the contact information at the bottom of this document. The user should be prepared to either send their data files or receive vague responses about what could be causing the problem.

**Contact Information**

The user may contact the creator of the program at the following email address.

[oschlimgen@gmail.com](mailto:oschlimgen@gmail.com)

Owen Schlimgen