Software Design and Specifications

URCVT v.1.2.0 02-NY Software Design and Specifications v.1.0.0 document is solely for use in the State of New York. This document can be expanded or updated as is necessary or required. Any recommendations listed in this document should not supersede user jurisdiction procedures or other controlling governance entities.

URCVT v.1.2.0 02-NY Software Design and Specifications v.1.0.0

Coding Standards and Style

The URCVT adheres to all the guidelines and requirements in the VVSG Software Requirements. We use *Google Checkstyle for Java* as our published, reviewed, and industry-accepted code style. For details see: github.com/checkstyle/checkstyle/blob/master/src/main/resources/google_checks.xml.

Code development and review processes are described in the ***URCVT v.1.2.0 13-NY Quality Assurance Plan v.1.0.0*****.**

Java 14

The tabulator is written in Java because it meets the VVSG requirements for software language selection. It is widely supported and popular in both industry and the open-source community for a wide variety of applications. It offers a mature and robust collection of third-party libraries. The Java Runtime Environment is standard on our target platforms which means a simple installation process. Our specific version of Java is OpenJDK 14.0.1, downloaded from here.

Open Source

We develop the tabulator as an open-source project for three main reasons:
1) Transparency: published source code increases public confidence in the application by giving anyone the opportunity to review our work and the processes and methodology behind it.
2) Adoption: open source licensing encourages others to use the software to facilitate the spread of ranked-choice voting.
3) Collaboration: open source licensing enables other software developers to contribute enhancements to the project and incorporate it into other related projects (RCV visualizers, policy research, etc..)

Architecture

The URCVT consists of one in-house java code module built from 23 source files. It relies on basic java platform libraries (file I/O, string processing, logging) and several 3rd-party modules listed below for reading and writing various file formats. These code modules are compiled and packaged with a minimal java runtime environment (14.0.1) which executes compiled object code, when installed and run on the target system.

3rd-party Modules

URCVT incorporates several 3rd-party modules which are all open-source. These meet the VVSG requirements for third-party modules. They are mature and widely accepted and used. None of them are modified in any way.

Module Name	Version	Purpose	Link
Apache Commons CSV	1.8	CSV "Comma Separated Values" reader / writer.	https://commons.apache.org/proper/commons-csv/user-guide.html
Apache POI OOXML	4.1.2	Excel spreadsheet reader / writer.	https://poi.apache.org/apidocs/dev/org/apache/poi/ooxml/package-summary.html
Jackson Core	2.11.1	XML / JSON streaming reader / writer core	https://github.com/FasterXML/jackson-core
Jackson Annotations	2.11.1	XML / JSON deserialization annotations	https://github.com/FasterXML/jackson-annotations
Jackson Databind	2.11.1	XML / JSON deserialization	https://github.com/FasterXML/jackson-databind
Jackson Dataformat XML	2.11.1	XML reader / writer	https://github.com/FasterXML/jackson-dataformat-xml
Jupiter JUnit API	5.6.2	Automated testing (used only during development)	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api
Jupiter JUnit Engine	5.6.2	Automated testing (used only during development)	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine

Software Limits:

Limitations of the Tabulation software i.e. how many cvrs can be tabulated are detailed in the ***URCVT v.1.2.0 03-NY System Hardware Specification v.1.0.0*** document.

Contest Tabulation Logic

Overview:
When the user triggers a contest tabulation, the application creates a new Tabulator Session object to manage the process flow. The Tabulator Session loads, parses, and validates the contest config file. If those steps succeed, it reads the cast vote records into memory, runs the tabulation, and generates the results summary files. Throughout these processes logging output is written to two locations as detailed in the Tabulator Logging Document.

1) Read the config file
2) Validate the config file
3) Read cvr files
4) Round by round tabulation of votes according to configuration
5) Generate reports

For detailed user guides, see ***URCVT v.1.2.0 08-NY System Operations Procedures v.1.0.0*** and ***URCVT v.1.2.0 300-NY Configuration File Parameters v.1.0.0***. Note that this document covers all tabulation logic in the URCVT in order to fully describe the software design.

Reading a config file

When a user selects an existing config file to open, the tabulator parses the JSON file and stores the information in a RawContestConfig object. This object is wrapped inside a ContestConfig object. The ContestConfig object serves as an interface between the config file and the rest of the application, providing extensive validation logic and a number of convenience methods for accessing normalized versions of the config settings.

A config file includes a tabulatorVersion string and a collection of parameters organized into four sections.

The tabulatorVersion is used to confirm that the version of the application that's loading the config file is compatible (specifically: not older than) the version of the application that created the file.

The outputSettings section contains settings related to what forms of output should be generated and where it should be saved on disk.

The cvrFileSources section is a list of one or more source file paths containing cast vote records, along with the parameters necessary for the tabulator to parse the records successfully.

The candidates section lists all of the candidate names/codes that appear in any of the CVR source files.

The rules section contains all of the parameters that determine exactly how the application should tabulate the cast vote records and produce results.

Writing the config file

A user can create a new config file or open an existing one and edit it. The GUI allows the user to modify all of the parameters that populate a config file (except for tabulatorVersion, which is determined by the app), run a validation to confirm that all of the settings in the file are valid, and save the file.

Validation

Config file validation checks each aspect of the file and attempts to verify that all of the individual settings are compatible with one another and should result in a successful tabulation. This includes confirming that the tabulation rules are consistent, that the candidate names are valid and don't contain duplicates, and that each source cvr file exists on disk, and has the proper parameters set for that cvr provider. The GUI prevents the user from creating a config file that would fail many of these checks but the validation assumes nothing and provides an additional layer of protection against user error. Because a user can create or modify a config file simply using a text editor the tabulator can't assume that a config file has only been modified by its own GUI.

Reading CVR Files

Reading the cast vote records into memory consists of parsing the source files and creating a CastVoteRecord object to represent each record. This object contains the candidate rankings for the CVR along with additional information parsed from the source file such as an ID and a precinct name. As the tabulation progresses, the CastVoteRecord object also stores information about the CVR's fate in each round (which candidate(s) its vote is counting toward and what fraction of the vote belongs to each). The details of parsing each source file depend on that file's provider.

Tabulation of CVRs

(Note: that NYC uses only single winner ranked-choice voting rules. Where referenced, multi winner tabulation rules will not be applicable.)

For the tabulation itself, the application creates a Tabulator object. This object's tabulate method runs a while loop that iterates until it determines that the tabulation is complete. Each iteration of the loop is a new round. Each round starts by calling computeTalliesForRound, which iterates over all of the cast vote records and sums up the total number of current votes for each candidate. This involves a number of steps:

1. If the CVR has already been marked as exhausted, it is skipped.
2. If the CVR was counted toward a candidate in the previous round and that candidate is still active, it's counted toward that candidate again in this round.
3. If the CVR has no rankings at all, it is marked as exhausted.
4. The tabulator then begins iterating through the rankings in the CVR, starting with the most preferred rank found (i.e. the lowest rank number, which is typically 1) and proceeding in order. At each rank, it checks for a number of possible cases:
   i. If the number of rankings skipped between the last ranking seen and this one exceeds the maxSkippedRanksAllowed value in the config, the CVR is marked as exhausted.
   ii. If one or more of the candidates at this rank already appeared at another rank and the config has enabled exhaustOnDuplicateCandidate, the CVR is marked as exhausted.
   iii. If the rankings at this rank constitute an overvote, the CVR is handled according to the overvote rule set in the config.
   iv. If a continuing candidate is found at this rank, the CVR is marked as counting toward the candidate.
   v. Otherwise, the CVR is marked as exhausted.
5. If the config has enabled tabulation by precinct, the method also updates the per-precinct tallies for this round.

Next, if the tabulation is in the first round and/or the contest is for a single seat, the software sets/updates the winning threshold (the number of votes that a candidate must meet or exceed in order to be named a winner) based on the winning rules set in the config.

The software then checks whether there are any continuing candidates with vote tallies in the current round that meet or exceed the winning threshold. Each of these candidates is marked as a winner and, if the winning rules in the config indicate that surplus votes should be redistributed (which is the case when the number of seats is greater than one and the winning mode is not MULTI_SEAT_BOTTOMS_UP_UNTIL_N_WINNERS), then the software calculates a surplus fraction and updates each CastVoteRecord object currently counting toward the winning candidate to record the portion of that vote that should remain allocated to the candidate in future rounds.

Finally, if no winners have been identified in the current round, the software determines whether it should identify one or more candidates to eliminate. (This is true if a) the number of identified winners is fewer than the number of seats for the contest, b) there are more than two candidate remaining and the winning rules specify a single-seat contest that should continue until only two candidates remain, or c) the winning mode is MULTI_SEAT_BOTTOMS_UP_USING_PERCENTAGE_THRESHOLD.) In this case, the candidate(s) to be eliminated are identified by trying the following methods in order until one of them returns one or more candidates:

1. If the cast vote records include undeclared write-ins and this set of candidates have not been eliminated yet, select them.
2. If the config specifies a minimum vote threshold for a viable candidate and one or more continuing candidates has a tally below that threshold, select them.
3. If the config enables batch elimination, attempt to identify two or more candidates who can be eliminated via this process.
4. Otherwise, select the remaining candidate with the lowest tally. (If multiple candidates are tied for the lowest tally, select one according to the tie-breaking rule specified in the config.)

The tabulator then determines whether it should continue to the next round and repeat the process. It continues if one of the following conditions is true:

a. For a single-seat contest, one of these is true:
    i. A winner has not been identified yet.
    ii. “Continue until two candidates remain” is enabled and one of these is true:
        1. There are more than two candidates remaining.
        2. The eliminations that reduced the number of remaining candidates to two happened in the current round. (This condition is included to ensure that the tabulator will generate an additional round showing the final vote tallies when only the last two candidates remain.)
b. For a multi-seat contest, one of these is true:
    i. The winning mode is MULTI_SEAT_BOTTOMS_UP_USING_PERCENTAGE_THRESHOLD and no winners have been identified yet.
    ii. The winning mode is not MULTI_SEAT_BOTTOMS_UP_USING_PERCENTAGE_THRESHOLD and the number of winners identified is fewer than the number of seats.
    iii. The winning mode is neither MULTI_SEAT_BOTTOMS_UP_USING_PERCENTAGE_THRESHOLD nor MULTI_SEAT_BOTTOMS_UP_UNTIL_N_WINNERS and the number of winners equals the number of seats, but the final winners were identified in the current round. (Similar to above, this condition ensures that the tabulator will generate an additional round showing the final surplus redistribution.)

When the tabulator determines that it should not continue tabulating, the tabulation is complete.

Reporting results

After the tabulation completes, the software generates results files and saves them to disk. These results are based on the following data that the tabulation process has produced and stored in memory:

- Which round each eliminated candidate was eliminated
- Which round each winning candidate was identified as a winner
- The winning threshold that was used to select the winner(s)
- Each candidate's vote tally in each round
- If the config has enabled tabulating by precinct, each candidate's vote tally in each precinct in each round
- A record of the number of votes in each round that were transferred from each candidate to each other candidate (or were exhausted)
- In a multi-seat contest involving surplus redistribution, the cumulative amount of residual surplus in each round
- The number of exhausted ballots in each round, which are the number of ballots that rank no continuing candidates - those candidates who are still active in the contest. This ranked-choice voting specific category of ballots includes undervoted ballots and overvoted ballots. Exhausted ballots are referred to as inactive ballots in summary results files.

Using this data, the tabulator creates a ResultsWriter object that writes two files to disk:

1. A CSV spreadsheet that includes the round-by-round tally for each candidate, when each candidate won or was eliminated, and related information
2. A JSON file that includes the same information found in the CSV spreadsheet, plus the number of votes transferred from each candidate to each other candidate (or exhaustion) in each round

If tabulating by precinct is enabled, the ResultsWriter also generates a CSV spreadsheet with round-by-round tallies for each precinct found in the cast vote records.

Finally, if the config file specifies that the tabulator should generate CDF (Common Data Format) output, it saves a CDF file in JSON format.

URCVT Logging

In addition to results files the URCVT generates various log outputs which describe:
- Program status

- Operations in progress
- Critical errors and warnings
- Tabulation-specific data: e.g. config file contents and how each vote counted in each round
- Additional system information

Log output is logically divided into two data streams:

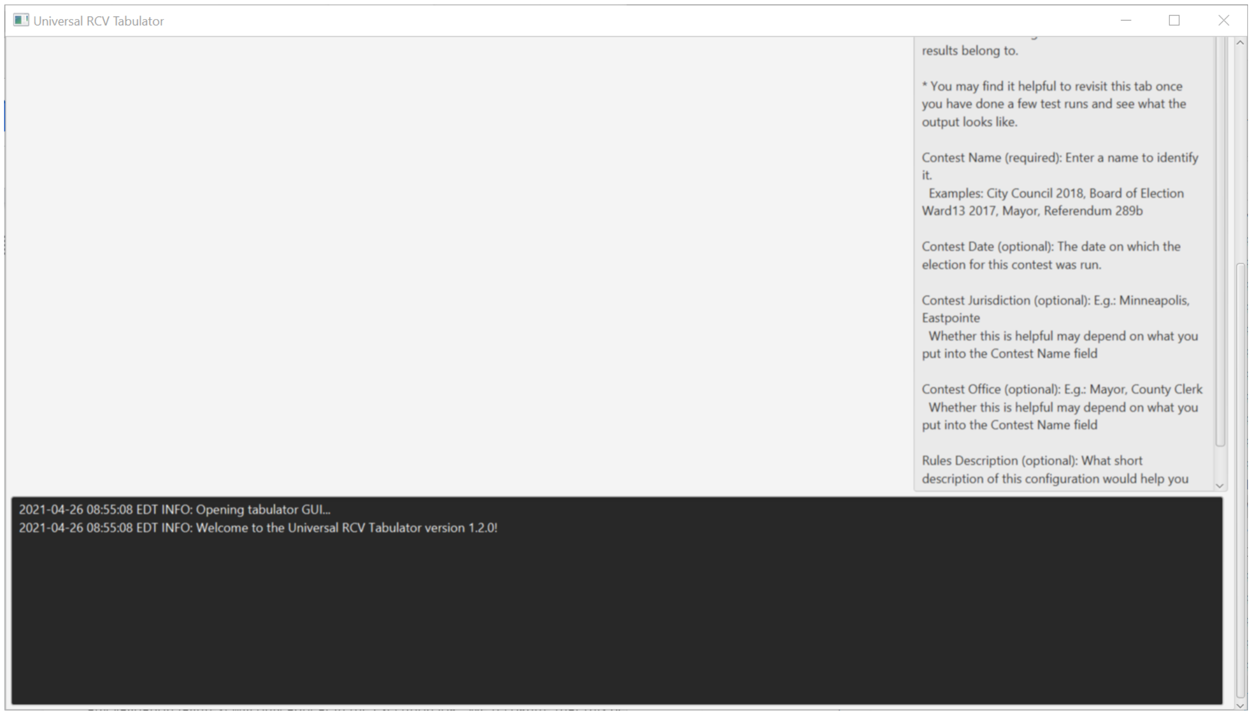
Stream 1: Tabulator "Operator" Logging

This data stream captures the overall operation of the Tabulator which may include: loading, editing, validating, and saving multiple config files, running cvr conversion functions, and tabulating multiple contests.

File Name: rcv_0.log
File Rotation:
When rcv_0.log reaches 50MB size it will be renamed along with any other preceding log files.
For example:
rcv_0.log -> rcv_1.log
rcv_1.log -> rcv_2.log
Then a new rcv_0.log file will be created and logging will continue.
This is a standard log rotation strategy that limits log file sizes for easier management.

File location: The Tabulator Operator Log will be created in the current working directory. When launched using the rcv.bat file this is next to the rcv.bat file. For example:
C:\Users\MyUser\rcv\bin\rcv_0.log

Operator logging is duplicated in the black GUI console box at the bottom of the application window. The console is provided as a convenience primarily for showing validation errors and providing feedback to the user. See screenshot below for example of this console box.



Universal RCV Tabulator

results belong to.

* You may find it helpful to revisit this tab once you have done a few test runs and see what the output looks like.

Contest Name (required): Enter a name to identify it.
Examples: City Council 2018, Board of Election Ward13 2017, Mayor, Referendum 289b

Contest Date (optional): The date on which the election for this contest was run.

Contest Jurisdiction (optional): E.g.: Minneapolis, Eastpointe
Whether this is helpful may depend on what you put into the Contest Name field

Contest Office (optional): E.g.: Mayor, County Clerk
Whether this is helpful may depend on what you put into the Contest Name field

Rules Description (optional): What short description of this configuration would help you

```
2021-04-26 08:55:08 EDT INFO: Opening tabulator GUI...
2021-04-26 08:55:08 EDT INFO: Welcome to the Universal RCV Tabulator version 1.2.0!
```

Stream 2: Contest-Specific "Audit" Logging

This data stream captures information about a specific contest tabulation. It includes all the same information written to the Execution Log (within the context of a single contest tabulation), and includes a listing of the config file being tabulated, and a record of how each cvr was counted in each round.

This data stream only begins logging after a config file has been validated. Thus all config validation logging (including any validation failures) will only appear in the execution log. We recognize that this behavior is non-intuitive and we have filed an issue to improve it: https://github.com/BrightSpots/rcv/issues/125

File Name: [time_stamp]_audit_0.log where timestamp is created when the tabulation is triggered and used on all output files for a given tabulation. For example: 2021-04-24_22-49-49_audit_0.log
File Rotation uses the same strategy as the Execution Logging. When an audit log reaches 50MB size it will be renamed along with any other preceding log files. For example:
 2021-04-24_22-49-49_audit_0.log -> 2021-04-24_22-49-49_audit_1.log
 2021-04-24_22-49-49_audit_1.log -> 2021-04-24_22-49-49_audit_2.log
Then a new 2021-04-24_22-49-49_audit_0.log file will be created and logging will continue.

File Folder Location is specified in the config file under "outputDirectory"

Interface:
The tabulator was originally developed for a command-line interface. The GUI (graphical user interface) was introduced both to make the process of configuring and running a tabulation faster and more intuitive and to enable users with a less technical background to use the software. The command-line interface still exists as a way to support execution via script, e.g. for batch processing and test automation. The GUI prevents the user from creating a config file that would fail many of these checks. A brief user guide to the Command Line Interface is available in ***URCVT v.1.2.0 240-NY Command Line Instructions v.1.0.0***. More information about error messages can be found in ***URCVT v.1.2.0 430-NY Universal RCV Tabulator Operator Log Messages v.1.0.0***.

Supported File types:
The Tabulator uses the JSON format for contest configuration files and one style of results summary output. JSON is simple, popular, and easy for humans and software to read and write. The Tabulator uses CSV (comma-separated values) for the tabular version of its results summary output. CSV is a non-proprietary format that all modern spreadsheet applications can recognize. Tabulation output log files are produced in plain-text with a .log extension for clarity. The Tabulator reads .xlsx (Microsoft Excel) and .xml (Extensible Markup Language) cvr and elections metadata files supplied by various vendors (ES&S, Hart, CDF, Unisyn).

Additionally the following documents were used to design the Universal RCV Tabulator software:

URCVT v.1.2.0 110-NY Tabulation Options for RCV Tabulation v.1.0.0
URCVT v.1.2.0 120-NY Process Ranked Choice Voting Contest v.1.0.0
URCVT v.1.2.0 130-NY Expected Outcome RCV Test Sets Single-Winner v.1.0.0
URCVT v.1.2.0 140-NY Expected Outcome RCV Test Sets Multi-Winner v.1.0.0
URCVT v.1.2.0 150-NY ES&S Ballot Limitations & Maximum Testing Range v.1.0.0
URCVT v.1.2.0 07-NY System Security Specification Requirements v.1.0.0

Document Revision History

Date	Version	Description	Author
04/27/2021	1.0.0	Software Design Specifications	Louis Eisenberg