Votetest: Voting system logic testing for VVSG 1.1

Draft

2009-06-22 15:17

Legal notices

Test materials disclaimer

This document and associated files have been prepared by the National Institute of Standards and Technology (NIST) and represent draft test materials for the Election Assistance Commission's next iteration of the VVSG. It is a preliminary draft and does not represent a consensus view or recommendation from NIST, nor does it represent any policy positions of NIST.

Software disclaimer

This software was developed at the National Institute of Standards and Technology by employees of the Federal Government in the course of their official duties. Pursuant to Title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain. This software is an experimental system. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. We would appreciate acknowledgement if the software is used.

This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified.

Commercial products disclaimer

Specific computer hardware and software products are identified in this paper to support reproducibility of results. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

Contents

List of Figures

List of Tables

1 Introduction

1.1 What Votetest Is

The Votetest distribution, or simply "Votetest" for short, is a package of public domain data, software, and documentation that the National Institute of Standards and Technology (NIST) is developing. Its purpose is to provide conformance test materials for use by Voting System Testing Laboratories. These test materials are used to assess conformity to the Voluntary Voting System Guidelines (VVSG) [1].

Votetest defines abstract tests that exercise every phase of the voting process from election definition through report generation. The scope of functionality that they cover to some extent is therefore quite broad. The abstract tests are "realized" according to the specifics of the voting system being tested.

In addition to the abstract tests, Votetest includes the expected results for each test, a means to derive the expected results for newly developed tests, a test generator, and detailed documentation.

1.2 What Votetest Is Not

Votetest CANNOT simply be plugged into a voting system to obtain a verdict on conformity. The tests that are defined abstractly by Votetest must be translated into concrete tests that can actually be run on a given voting system. This task requires the same expertise and diligence that test labs have employed in the past when developing their own tests from scratch.

Votetest, by itself, DOES NOT provide complete coverage of the VVSG. Votetest is only one tool that is used in one part of the conformity assessment process. It is designed not to be used in isolation, but rather to complement the other testing activities, which include the physical configuration audit, documentation and design reviews, electromagnetic compatibility and environmental testing, security reviews, usability and accessibility assessments, and the evaluation of reliability, accuracy, and misfeed rates. Test labs should consult Volume II of the VVSG [1] regarding the full scope of testing to be performed.

Votetest DOES NOT address "fitness for use." Its scope is strictly limited to conformity assessment. It is not a substitute for Voting System Testing Laboratories' diligent assessments of the full scope of voting system functionality (including vendor-specific functionality), nor does it obviate the need for acceptance testing by jurisdictions.

1.3 Structure of this document

Section 2 provides additional background of general interest. Section 3 provides detailed technical documentation on the portions of Votetest that are essential to performing conformity assessment. Section 4 and Section 5 provide detailed technical documentation on the portions of Votetest that could be used to develop additional test cases. Finally, Section 6 walks through the process of defining a new test case based on an example election.

2 Background

By authorization of the 2002 Help America Vote Act (HAVA), NIST is assisting the Election Assistance Commission (EAC) with the implementation of Voluntary Voting System Guidelines (VVSG) for states and local governments conducting Federal elections. The EAC's Technical Guidelines Development Committee (TGDC) in collaboration with NIST researchers has developed a draft of Version 1.1 of the VVSG. The draft document is a set of detailed technical requirements addressing core requirements, human factors, privacy, security, and transparency of the next generation of voting systems. The EAC plans to issue the final Version 1.1 after receiving and reviewing public comments.

NIST is developing a set of uniform public test suites to be used as part of the EAC's Testing and Certification Program. Test labs will be able to use these freely available test suites to help determine that VVSG requirements are met by voting systems. The test suites address human factors, security and core functionality requirements for voting systems as specified in the VVSG. Use of the public test suites will produce consistent results and promote transparency of the testing process. The test suites can also assist manufacturers in the development of conforming products by providing precise test specifications. Also, they will help reduce the cost of testing since each test lab would no longer need to develop its own test suites. Finally, a uniform set of public test suites can increase election officials' and voters' confidence that voting systems conform to VVSG requirements.

3 Test materials

3.1 Overview of test materials

This is the documentation for the test materials portion of the Votetest distribution. Its intended audience is test labs accredited by the Election Assistance Commission to perform voting system certification testing. The test lab is assumed to have competence with the following:

- The Voluntary Voting System Guidelines (VVSG) [1];
- Data modelling;
- Structured Query Language (SQL) [2].

Votetest includes the following components:

- The data model (Section 3.4) documents the world view inherent in the schema and test suite. It is general enough to support arbitrary combinations of all of the voting variations defined in VVSG 2.0.
- The basic schema (Section 3.5) is an SQL realization of the data model.
- The basic test suite (Section 3.6) is a set of abstract test cases expressed in SQL using the basic schema. The test cases are delivered as separate files in the Votetest distribution. These tests trace primarily to the requirements of [1, Volume I Chapter 2]; traceability details will be provided in Section 3.8. However, because they exercise every phase of the voting process from election definition through report generation, the scope of functionality that they cover to some extent is quite broad.

- The expected results (Section 3.3.4.3) are text files providing the vote totals that a conforming voting system should produce for each test case.
- The advanced schema (Section 4) is an SQL realization of the logic model of VVSG 2.0 [3, Part 1 Section 8.3], which specifies the results that voting systems are required to report. [1] does not include a logic model to define precisely what values are expected to be reported for votes, overvotes, and undervotes; however, [3, Part 1 Section 8.3] is believed to be consistent with the intent of [1].
- The advanced test development environment (Section 5) includes the infrastructure and tooling used to develop test cases and determine their expected results.

While Votetest includes both basic and advanced materials, only the data model (Section 3.4), basic schema (Section 3.5), basic test suite (Section 3.6) and expected results (Section 3.3.4.3) are essential to the conformance testing process. Section 3.3 explains in more detail how Votetest is used in that process.

3.2 File listing

The Votetest distribution is provided as a zip file named votetest-*NNN*-*YYYYMMDD*.zip and alternately as a compressed tar file named votetest-*NNN*-*YYYYMMDD*.tar.bz2, where *NNN* is an identifying number for the VVSG document referenced (see Table 1) and *YYYYMMDD* is a sequence of digits indicating the date of the release. The files contained in the distribution are described in Table 2.

Table 1: Identifying numbers for VVSG documents

NNN	**Reference**
100	VVSG 1.0 final 2006-03-06 [4]
101	VVSG 1.1 draft 2009-05-27 [1]
190	VVSG 2.0 draft 2007-08-31 [3]
none	VVSG 2.0 draft 2007-08-31 [3]

3.3 How to use Votetest

The steps listed in the following subsections fit within the context of the overall conformity assessment process that is described in [1, Volume II Chapter 1]. Thus, in practice, the steps will not necessarily follow directly one to the next as is implied here, because there may be other conformity assessment activities occurring in between or parallel to them.

3.3.1 Determine relevant voting variations—system level

The Technical Data Package received from the manufacturer is required to include an *implementation statement*, which is specified by the Conformance Clause of the VVSG [1, Volume I Section 1.5.2]. This implementation statement includes the manufacturer's classification of the voting system as a whole and of the different devices of which it is comprised. For Votetest, the relevant

Table 2: Basic files in the Votetest distribution

Files	Description	Details
disclaimer.txt	Text file containing the test materials disclaimer.	Legal notices
COPYING	Text file containing the software disclaimer.	Legal notices
doc/	Subdirectory containing the source and PDF of this documentation.	N/A
1-basic-*description*.sql	SQL, abstract test cases of the basic test suite (92 files).	Section 3.6
sample_output/	Subdirectory containing text files that provide the expected results for each test case and each integrity check (106 files).	Section 3.3.4.3
sample_output_kill-overvotes/	Same, but with overvoting suppressed (106 files).	Section 3.3.4.3
ChangeLog	Text file containing the change log for the Votetest distribution.	N/A
... *other files* ...	Files pertaining to the advanced test development environment.	Table 8

voting variations are determined by examining the implementation statement. The completeness and correctness of the manufacturer's classification should have been established in a preceding documentation and design review.

The list of voting variations identified in Votetest differs from the list provided in [1, Volume I Section 2.1.7.2]. Table 3 explains the differences.

At this stage, what is relevant is the system-level classification. If the system as a whole does not support a particular voting variation, then support for that variation in isolated devices is irrelevant.

3.3.2 Determine applicable tests

Table 4 (coming up in Section 3.6) lists the tests in the basic Votetest test suite and indicates the voting variations that are used by each test. The set of tests applicable to a given voting system is the set of all tests that **do not use** any voting variations that the voting system **does not support**. In other words, every test is applicable unless it is specifically excluded, and a test is excluded only if it uses a voting variation that, according to the reviewed and accepted implementation statement, the system does not implement.

For example, if a voting system *did not* support ranked order voting, the following tests would be excluded:

1-basic-NoBallots-RankedOrder.sql
1-basic-RankedOrder-1.sql
1-basic-RankedOrder-2.sql
1-basic-StraightParty-RankedOrder.sql
1-basic-Primary-RankedOrder.sql
1-basic-BallotRotation-RankedOrder.sql

Table 3: List of voting variations, Votetest vs. the VVSG

Votetest	**VVSG**	**Explanation**
Primary elections	Closed primaries Open primaries	See Section 3.6.
Absentee voting Absentee by categories	N/A	Absentee voting is not on the list in Volume I Section 2.1.7.2. There is more than one conforming approach.
Split precincts	Split precincts	Same.
Ballot rotation	Ballot rotation	Same.
Write-ins	Write-in voting	Same.
Cumulative voting	Cumulative voting	Same.
N-of-M voting	Vote for N of M	Same.
Ranked order voting	Ranked order voting	Note: Votetest assumes that support for ranked order voting includes support for multiple-winner ranked order contests.
Provisional/challenged ballots	Provisional or challenged ballots	Same.
Straight party voting	Straight party voting	Same.
Cross-party endorsement	Cross-party endorsement	Same.
N/A	Partisan offices Non-partisan offices Primary presidential delegation nominations Recall issues, with options	In the drafting of VVSG 2.0, these were all found to be subsumed by other voting variations with appropriately configured ballot text.

1-basic-AbsenteeByCategory-RankedOrder.sql
1-basic-Cumulative-RankedOrder.sql
1-basic-SplitPrecinct-RankedOrder.sql
1-basic-RankedOrder-WriteIns.sql
1-basic-RankedOrder-NofM.sql
1-basic-RankedOrder-Provisional.sql
1-basic-AbsenteeBySpecialPrecinct-RankedOrder.sql

There is a special case regarding absentee voting that will be addressed in Section 3.6.

3.3.3 Multiply tests for different interfaces and devices

At this point the test lab has the beginnings of a test plan with a list of the tests that are applicable to the voting system as a whole. That plan must now be expanded down to the device level.

When a voting system offers multiple paths by which voting can occur, such as a touchscreen interface and an audio ballot on one device plus optical scanning of paper ballots by another device, each of these paths must be tested. It is entirely possible for a failure to be specific to a particular user interface. For example, some controls that can be activated using a touchscreen might be incorrectly skipped over if an input device with a navigational model (next option, previous option) is used.

Several different strategies could be used to divide or replicate the testing among the available interfaces and devices. However, since most tests in the basic test suite are small by design, with relatively few ballots, the recommended approach is to repeat each test on each device and interface separately. For large tests such as the VVSG 2.0 volume test (mock election), one would instead divide the ballots among the available devices and interfaces using a distribution that imitates a realistic election.

Absentee voting is again a special case. In systems where absentee ballots follow a separate path from other ballots (e.g., a dedicated central count optical scanner), it makes sense to repeat tests over the absentee interface. However, the basic test suite also contains tests specifically designated for absentee voting, that mix absentee ballots with non-absentee ballots. These designated tests would be executed using both the absentee and non-absentee paths at the same time, not repeated on each path separately.

It is not necessarily the case that every device in the system would support every voting variation claimed at the system level. Thus there may be cases where it is not possible to multiply certain tests onto certain devices. However, other devices that do pass those tests must be present to enable satisfaction of the system-level claim.

3.3.4 Perform testing

Once the test plan has been approved, work on implementing the plan can begin. The steps that the test lab must do for each applicable Votetest test are summarized in Figure 1 and explained in more detail in the following subsections.

3.3.4.1 Translate test from Votetest model to voting system

Since interfaces to voting systems vary widely, the test lab must use the basic test suite as input from which to generate system-specific tests. For each applicable test, the test lab translates the abstract, SQL version included in the basic test suite (Section 3.6) into a concrete test for the voting system under test, thereby "realizing" the test. The data model (Section 3.4) and basic schema (Section 3.5) are essential to understanding the abstract tests so that they can be realized correctly.

Figure 2 shows an example of a Votetest test expressed in SQL. It specifies both the election definition and the ballots and votes to be cast.

Things to be determined for the voting system under test would include:

- Initial state: Determine how the voting system should be set and reset to a starting state prior to each test.
- Election definition:
  - Read the Voting Equipment User Documentation supplied by the manufacturer to understand how election definition is performed using the Election Management System.
  - Determine how the definitions of the ballot styles, contests, choices, and reporting contexts used in the Votetest basic test suite would be implemented in the voting system under test.

  Figure 3 shows an example of entering part of the election definition from Figure 2 into a hypothetical voting system. In an actual voting system, entering the complete election definition might require many separate interactive steps, such as creating the election, enumerating political parties, creating precincts, creating contests, creating choices within those contests, adding ballot text, setting options, etc. Additionally, the test lab may need to synthesize content for system-specific details that are not specified by Votetest. This example only shows the creation of a contest.
- Voting: Determine how the pattern of votes used in each test would be cast in the voting system under test. The system may support numerous different voting interfaces, each of which would have a different process for casting votes. For example, if a DRE is to be tested, scripts for "test voters" to follow must be produced. The scripts instruct the test voters on what votes to cast using the electronic interface of the DRE. If an optical scanner is to be tested, physical test ballots must be produced. This might be accomplished by handing a similar script to test voters to instruct them on how to fill in the paper ballots, or the test engineer might simply mark the ballots personally.

  Figure 4 shows an example of how the ballot with ID 2 in Figure 2 would translate for the precinct count optical scan interface of the hypothetical voting system. Figure 5 shows an example of how the same ballot would translate for the DRE interface of the hypothetical voting system.
- Reporting: Determine how to generate pre-election audit reports, system readiness audit reports, in-process audit reports, and vote data reports (i.e., reports of ballot counts and vote totals) in the voting system under test.

Figure 1: How to use a test

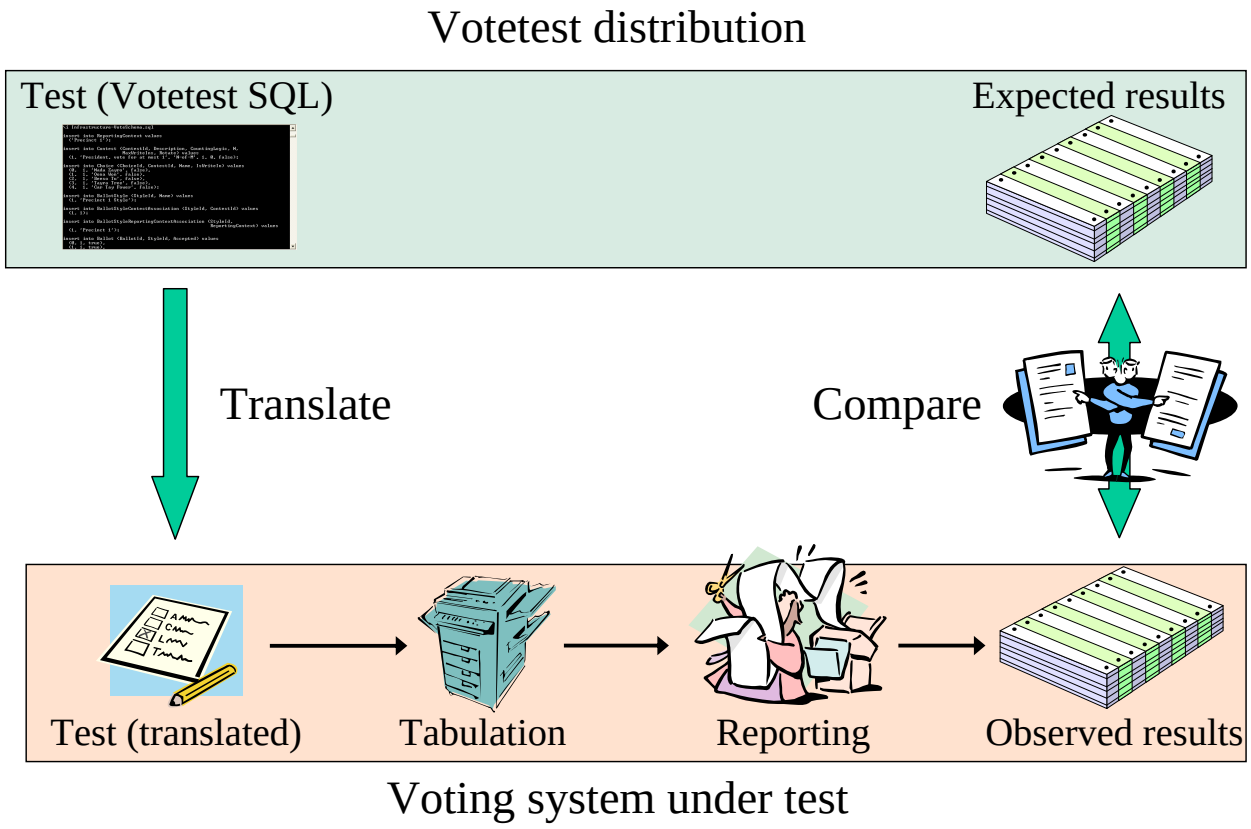


Figure 2: Example Votetest test

```
\i Infrastructure-TestHeader.sql

\echo '$Id: 1-basic-NofM.sql 452 2008-01-09 14:24:56Z dflater $'
\echo
\echo 'Small 2-of-M contest, no write-ins, no rejected ballots.'
\echo 'Ballot styles:  1'
\echo 'Reporting contexts:  1'

\i Infrastructure-VoteSchema.sql

insert into ReportingContext values
  ('Precinct 1');

insert into Contest (ContestId, Description, CountingLogic, N,
                     MaxWriteIns, Rotate) values
  (1, 'Parking Committee, vote for at most 2', 'N-of-M', 2, 0, false);

insert into Choice (ChoiceId, ContestId, Name, IsWriteIn) values
  (0,  1, 'Nada Zayro', false),
  (1,  1, 'Oona Won', false),
  (2,  1, 'Beeso Tu', false),
  (3,  1, 'Tayra Tree', false),
  (4,  1, 'Car Tay Fower', false);

insert into BallotStyle (StyleId, Name) values
  (1, 'Precinct 1 Style');

insert into BallotStyleContestAssociation (StyleId, ContestId) values
  (1, 1);

insert into BallotStyleReportingContextAssociation (StyleId,
                                                    ReportingContext) values
  (1, 'Precinct 1');

insert into Ballot (BallotId, StyleId, Accepted) values
  (0, 1, true),
  (1, 1, true),
  (2, 1, true),
  (3, 1, true),
  (4, 1, true),
  (5, 1, true),
  (6, 1, true),
  (7, 1, true),
  (8, 1, true);

insert into VoterInput (BallotId, ChoiceId, Value) values
  (1, 1, 1),
  (2, 2, 1),
  (2, 3, 1),
  (3, 2, 1),
  (3, 3, 1),
  (4, 3, 1),
  (4, 4, 1),
  (5, 4, 1),
  (6, 4, 1),
  (7, 4, 1),
  (8, 0, 1),
  (8, 1, 1),
  (8, 2, 1);

\i Infrastructure-TestHook.sql
\i Infrastructure-IntegrityChecks.sql
\! ReportGenerator/ReportGenerator "Precinct 1"
\i Infrastructure-TestFooter.sql
```

← Preamble

Election definition

Ballots

← Conclusion

Figure 3: Example EMS interaction

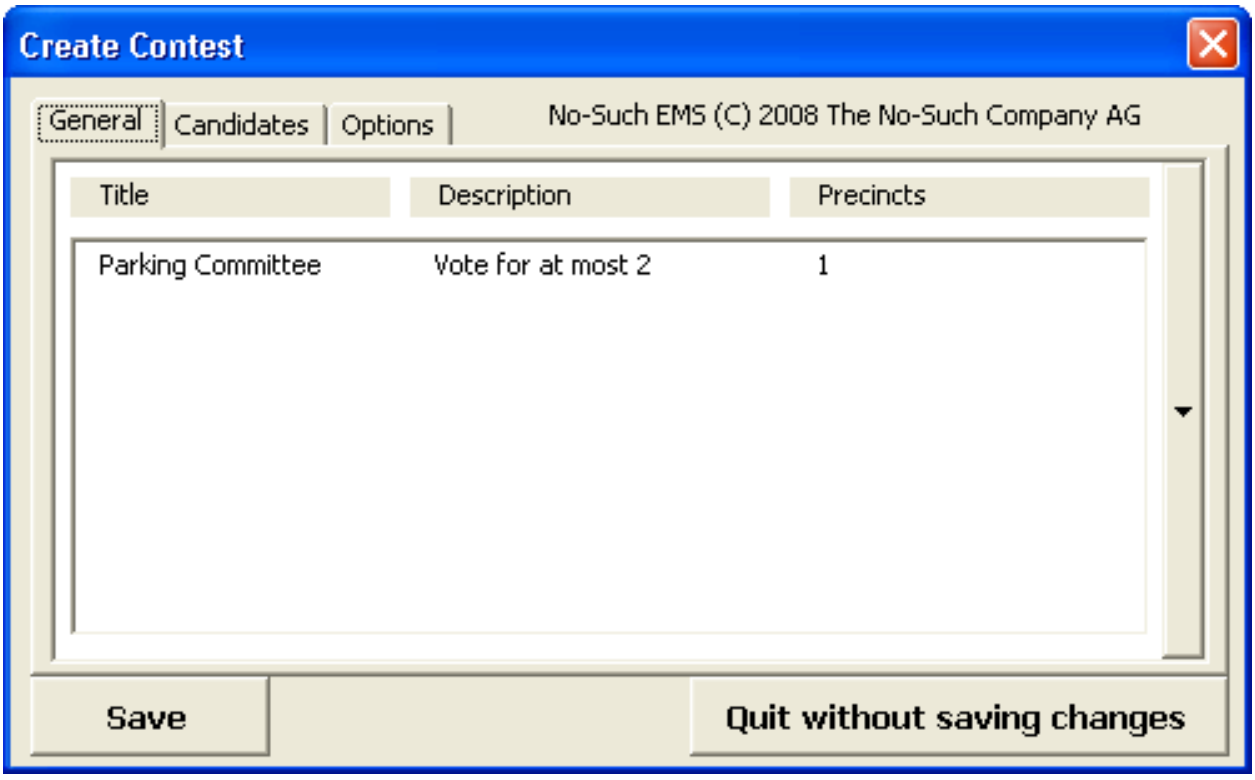


Figure 4: Example optical scan ballot

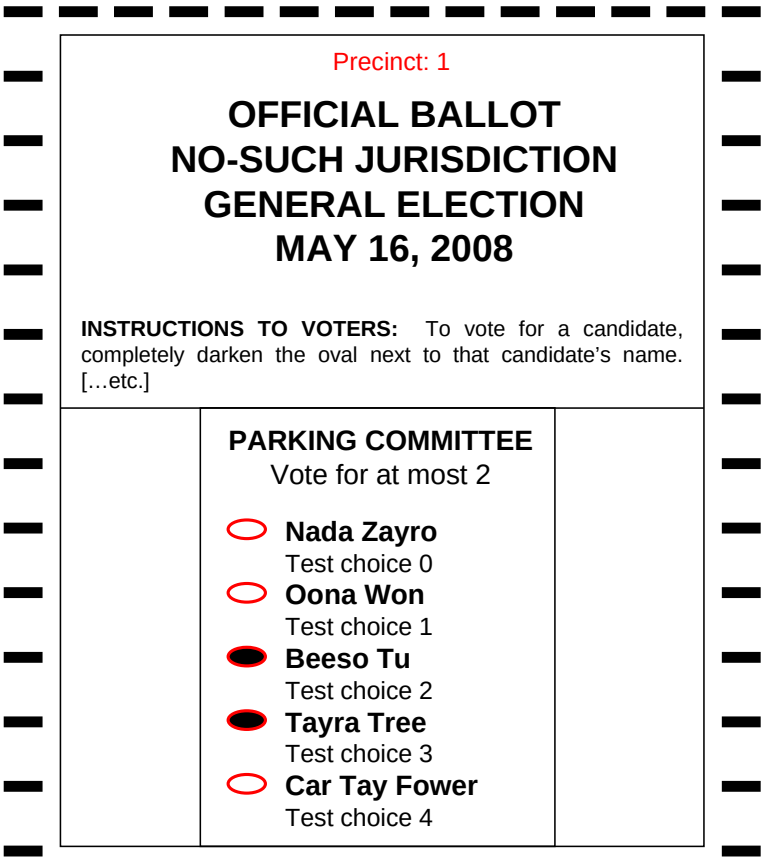


Figure 5: Example DRE voting script

*If at any step you make a mistake performing the **Action for tester**, go directly to the Botched Test Checklist.*

*If at any step the DRE does not respond as indicated under **Expected DRE response**, go directly to the Anomaly Checklist.*

Action for tester	**Expected DRE response**
1. Insert the card that the poll worker gave you into the slot on the front of the DRE.	The DRE should show the text "Ballot activated" followed by instructions to voters.
2. Touch NEXT on the touch-screen.	The DRE should advance to the PARKING COMMITTEE contest.
3. Touch the box to the left of **Beeso Tu**.	The box should turn green.
4. Touch the box to the left of **Tayra Tree**.	The box should turn green.
5. Touch NEXT on the touch-screen.	The DRE should advance to the summary screen.
6. Touch CAST BALLOT on the touch-screen.	The DRE should display "Please remove the card and deposit it in the shoe box near the exit." and "Thank you for voting."
7. Remove the card.	The DRE should return to its welcome screen, which shows the text "Precinct 1" and "Please insert card."

3.3.4.2 Execute test

For each test case, the test lab executes the "realized" test case within the voting system under test using the following general test template:

1. Establish initial state (clean out data from previous tests, verify resident software/firmware);
2. Program election and prepare ballots and/or ballot styles;
3. Generate pre-election audit reports;
4. Configure voting devices;
5. Run system readiness tests;
6. Generate system readiness audit reports;
7. Precinct count only:
    1. Open poll;
    2. Run precinct count test ballots; and
    3. Close poll.
4. Run central count test ballots (central count / absentee ballots only);
5. Generate in-process audit reports;
6. Generate data reports for the specified reporting contexts;
7. Inspect ballot counters; and
8. Inspect reports.

This template essentially puts the voting system through the technical paces of a very small election for each test. Although the election definition is simple and the number of ballots small, the process from beginning to end exercises all of the voting system functions that would be used in an election. The surrounding procedures and environment are, of course, quite different, and vastly simplified in the laboratory testing environment, so it is not a mock election in the sense that the VVSG 2.0 volume test is.

Also see [1, Volume II Section 1.8.2] regarding the test conditions, test fixtures, test data requirements, and test practices that apply.

3.3.4.3 Compare observed and expected results

As part of the test execution, the voting system under test is instructed to generate vote data reports. These contain the "observed results" with respect to ballot counts and vote totals. Figure 6 shows an example report from the hypothetical voting system.

The "expected results" for all of the test cases are saved as text files in two subdirectories of the Votetest distribution. The subdirectory sample_output contains the results expected from the test cases as written, while the subdirectory sample_output_kill-overvotes contains the results expected from the test cases with overvotes changed to undervotes.

Each results file has the following organization:

- Test header with timestamp and description of the test case.
- Some details that are not relevant to conformity assessment:
  - Results of data integrity checks. These are checks of the integrity of the test cases themselves and do not correspond to requirements on voting systems.
  - View materialization log.
- Reports, for one or more ReportingContexts, that include the vote totals that voting systems are expected to produce.
- Test footer including the report total volume and timestamp.

An example is shown in Figure 7. Comparing the "observed results" in Figure 6 and the "expected results" in Figure 7, it is clear that the quantities reported for candidate vote totals are identical, even though the form of the reports is somewhat different. The test lab would find no discrepancy in those results. However, the observed results shown in Figure 6 are not in and of themselves sufficient to satisfy all of the VVSG requirements on reporting. For example, [1, Volume I Sections 2.4.3 and 5.4.4] specify additional information that the voting system under test must provide in some report if the system is to be found conforming.

The other reports generated as part of the test (pre-election reports, etc.) would also be inspected to look for anomalies and to assess conformity to the L&A testing and auditability requirements of the VVSG. Similarly, any unexpected behavior of the voting system observed by the tester would be assessed by the resident expert (possibly but not necessarily the same person) to determine if a nonconformity was demonstrated.

Some notes and cautions apply:

1. Many test cases contain overvotes. However, in some voting systems, overvoting is prevented. Votetest handles this possibility through test transformation. For each ballot that overvotes a particular contest, an SQL script (Infrastructure-KillOvervotes.sql) deletes that ballot's votes in that contest, with the effect that the contest is undervoted instead of overvoted. Expected results are provided both for the test cases as originally written (in sample_output) and for the test cases transformed to remove overvotes (in sample_output_kill-overvotes).

2. For *Ballot rotation* tests, the test lab must check to make sure that the voting system does in fact produce and utilize rotated ballots as specified in the VVSG. The assignment of ballot choices to specific ballot positions is abstracted out of the Votetest data model, so the rotation behavior is not represented in the "expected results."

3. Ranked order logic is not normative. The algorithm used to derive the "expected results" for ranked order contests is only one example of conforming behavior. This algorithm is not recommended or endorsed by the National Institute of Standards and Technology for use in elections and it is probably not the best algorithm available for the purpose. It is used in Votetest only to provide output for comparison in simple cases where the implementation-dependent details have no impact.

Figure 6: "Observed results" for example

NO-SUCH JURISDICTION
GENERAL ELECTION
MAY 16, 2008

REPORT GENERATED AT 2008-05-16 16:12:23

	Total	%	Election Day	Early	Provisional
Precincts Counted	1	100.00			
Registered Voters	9				
Ballots Cast / Turnout	9	100.00	9	0	0
PARKING COMMITTEE (Vote for at most 2)					
FOWER, CAR TAY	4	40.00	4	0	0
TREE, TAYRA	3	30.00	3	0	0
TU, BEESO	2	20.00	2	0	0
WON, OONA	1	10.00	1	0	0
ZAYRO, NADA	0	0.00	0	0	0
Write-Ins (combined)	0	0.00	0	0	0
Overvotes	2		2	0	0
Undervotes	6		6	0	0

Figure 7: "Expected results" for example

```
######################################################################

           BEGIN TEST CASE OUTPUT   2008-04-10 16:15:52-04

######################################################################

$Id: 1-basic-NofM.sql 452 2008-01-09 14:24:56Z dflater $

Small 2-of-M contest, no write-ins, no rejected ballots.
Ballot styles:  1
Reporting contexts:  1


$Id: Infrastructure-VoteSchema.sql 475 2008-02-21 20:49:51Z dflater $
$Id: Infrastructure-TestHook.sql 453 2008-01-09 14:38:46Z dflater $
$Id: Infrastructure-IntegrityChecks.sql 281 2007-07-19 15:32:59Z dflater $

----- Begin integrity check output -- All results should be empty -----

  Out Of Range Voter Inputs
 ballotid | choiceid | value
----------+----------+-------
(0 rows)

… other integrity checks deleted …

----- End integrity check output ---- All results should be empty -----

$Id: ReportGenerator.cc 460 2008-01-14 15:10:24Z dflater $
Materializing views (for large tests, this may take a while).
2008-04-10 16:15:51-0400 Materialization begun
2008-04-10 16:15:51-0400 FilteredContextChoiceAssociation
… other view materializations deleted …
2008-04-10 16:15:52-0400 Materialization finished

--------------------------------------------------------------------------------
Report for context Precinct 1 generated 2008-04-10 16:15:52-0400

                                BALLOT COUNTS

Configuration                                Read    Counted
-------------                                ----    -------
Total                                           9          9
                                  Blank         1          1
Precinct 1 Style                                9          9
                                  Blank         1          1

                                VOTE TOTALS

Parking Committee, vote for at most 2
Car Tay Fower                                   4
Tayra Tree                                      3
Beeso Tu                                        2
Oona Won                                        1
Nada Zayro                                      0
Overvotes                                       2
Undervotes                                      6
Counted ballots                                 9
Balance                                         0
--------------------------------------------------------------------------------

Report total volume: 67
  - Includes optional reporting of blank ballots.
  - Excludes separate reporting of ballots cast vs. read.

######################################################################

            END TEST CASE OUTPUT   2008-04-10 16:15:52-04

######################################################################
```

← Test header

← Integrity checks

← Materialization log

← Report

← Test footer

3.4 Data model

The data model for voting system core requirements is described in Figure 8 by a Unified Modeling Language (UML) class diagram [5]. Following sections explain the diagram.

3.4.1 Assumptions

All entities in this data model are implicitly scoped by an election. It is assumed that different elections are stored in different databases, and any reuse of definitions from one election to another is accomplished by copying over the relevant data.

This data model is constructed from an integrated, top-level viewpoint. In practice, different portions of the system will deal with only a portion of the data at any given time. It is expected that test labs will project and extract data from the integrated schema as needed to support these limited viewpoints for testing.

The results of tabulation and reporting are derived from the content of the data model, but those results are themselves outside the scope of the model.

3.4.2 POD (Plain Old Data) types

BallotCategory (enum) Arbitrary tag that may be applied to Ballots; e.g., Early, Regular, In-Person, Absentee, Provisional, Challenged, NotRegistered, WrongPrecinct, IneligibleVoter. Categories are jurisdiction-defined but are likely to include several classes of provisional.

Boolean Normal true/false data type.

ContestCountingLogic (enum) N-of-M, Cumulative, Ranked order, or Straight party selection. (1-of-M is a special case of N-of-M.) The tabulation logic for a straight party selection Contest is implicitly 1-of-M, but with side-effects for other Contests.

NaturalNumber Integer greater than zero.

Text Normal character string.

WholeNumber Integer greater than or equal to zero.

3.4.3 Classes

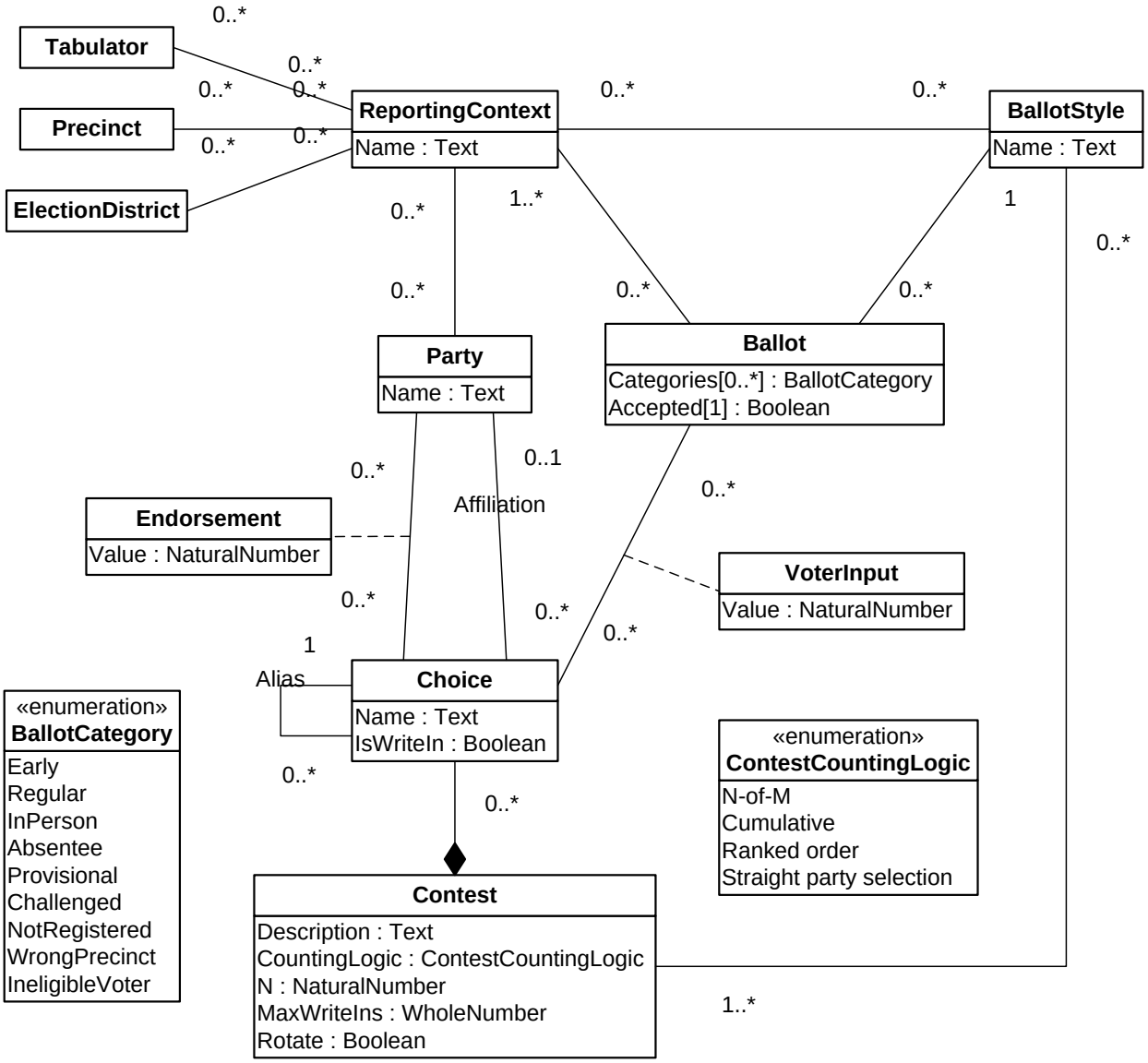
3.4.3.1 Ballot

The undefined primitive in all elections. The Contests that appear on a particular Ballot are defined by its BallotStyle. The applicable ReportingContexts include all those specified for its BallotStyle, but additional ReportingContexts may be specified for the individual Ballot.

Attributes of Ballot:

Categories Arbitrary, jurisdiction-defined tags applied to the Ballot; e.g., Early, Regular, InPerson, Absentee, Provisional, Challenged, NotRegistered, WrongPrecinct, IneligibleVoter.

Figure 8: Vote data model for core requirements



Accepted True if the Ballot should be counted, false if not (e.g., for a provisional Ballot that was not accepted).

3.4.3.2 BallotStyle

Set of Contests and ReportingContexts that is inherited by all Ballots of that style. Depending on the type of election and local practices, a jurisdiction would define a separate BallotStyle for each precinct, each split within a precinct, and/or for each political party.

Attributes of BallotStyle:

Name Human-readable identifier.

3.4.3.3 Choice

One of the things you can vote on in a Contest, such as a candidate, a political party, or yes or no. Choice is scoped by Contest, so even if the same person runs as a candidate in two or more Contests, those separate candidacies are represented by separate Choices. Choices do not map 1:1 with ballot positions—a Choice uniquely identifies a candidate, while a given ballot position might just be a generic write-in slot.

Attributes of Choice:

Name Human-readable identifier. (In a real system, Choices could have complex descriptive data associated with them that must be displayed to the user somehow, but for logic testing purposes a single field suffices.)

IsWriteIn True if the Choice is a write-in candidate, false if not.

3.4.3.4 Contest

Subdivision of a Ballot corresponding to a single question being put before the voters, consisting of header text, a discrete set of Choices, and possibly write-in opportunities. It is possible for a Contest to have zero Choices, e.g., if there are no registered candidates but write-ins are being accepted. Choices corresponding to the candidates written in would be added later.

Attributes of Contest:

Description Human-readable header text.

CountingLogic Identifies the tabulation method used for the Contest.

N For CountingLogic other than ranked order, N is the maximum number of votes that may be cast in the Contest by a given voter. In an N-of-M Contest, the voter may cast at most one vote for each Choice, so N is equal to the maximum number of Choices that the voter may select without overvoting.[1] In a cumulative Contest, there is no such constraint—the voter may cast more than one vote for a given Choice.

[1]The value of M, for N-of-M voting, is simply the number of Choices associated with the Contest and is not explicitly modelled.

Typically, N also is the number of winners for the Contest, but not necessarily. The voting system only needs to gather votes and report the totals; the picking of winners may be an external process impacted by election law, late-breaking judicial rulings, etc. However, for ranked order Contests, N *is* specifically the number of Choices to be elected, and has no other meaning.

MaxWriteIns The number of ballot positions allocated for write-ins; the maximum number of candidates that the voter may write in. Any value between zero and N is possible. Zero would mean that write-ins are not allowed; N would mean that write-ins are allowed; a number in between would mean that write-ins must be approved and the number of approved write-in candidates is less than N.

Rotate True if the ordering of Choices within the Contest should be rotated, false if not.

3.4.3.5 ElectionDistrict

Surrogate for real-world entity that may have associated ReportingContexts. Any relationship between ElectionDistricts and Contests appearing on the ballot in those districts is implemented by BallotStyles.

3.4.3.6 Party

Surrogate for real-world political party.

Attributes of Party:

Name Unique human-readable identifier.

3.4.3.7 Precinct

Surrogate for real-world entity that may have associated ReportingContexts. Any relationship between Precincts and Contests appearing on the ballot in those Precincts is implemented by BallotStyles.

3.4.3.8 ReportingContext

Particular scope within which the system must be capable of generating reports. E.g., to support reporting at the precinct level, there must be a ReportingContext for each precinct. The association between ReportingContexts and individual tabulators, precincts, election districts, political parties, ballot categories, or other arbitrary scopes of reporting is jurisdiction-defined and jurisdiction-managed, mostly using BallotStyles. The ways in which ReportingContexts overlap or include one another is entirely determined by the assignment of multiple ReportingContexts to BallotStyles and Ballots.

Attributes of ReportingContext:

Name Human-readable identifier.

3.4.3.9 Tabulator

Surrogate for real-world entity, e.g., a DRE or optical scanner, that may have associated ReportingContexts.

3.4.4 Named associations

3.4.4.1 Affiliation

Identifies the Party to which a candidate claims allegiance. Does not necessarily have anything to do with Endorsements.

3.4.4.2 Alias

Identifies an alternative Choice that for tabulation purposes is considered equivalent to a particular canonical Choice. Aliases will normally be variant spellings of a candidate's name that appeared in write-in positions.

3.4.4.3 Endorsement

Identifies a voter response that would be implied by a straight party vote for the endorsing Party. Does not necessarily have anything to do with Affiliation.

Attributes of Endorsement:

Value Analogous to VoterInput Value, this is the vote recommended by the endorser.

In a 1-of-M or N-of-M Contest, an Endorsement with Value = 1 would exist for the single Choice or for each of the Choices endorsed by the Party.

In a Cumulative Contest, Value may take on values greater than 1. For example, if the Party recommended that voters cast two votes for the first Choice and one vote for the second, an Endorsement with Value = 2 would exist for the first Choice and an Endorsement with Value = 1 would exist for the second Choice.

In a Ranked order Contest, Value contains the ranking that the Party recommends that voters assign to each Choice.

3.4.4.4 VoterInput

The response that a particular Ballot provides for a particular Choice.

Attributes of VoterInput:

Value The response of the voter in some ballot position. The absence of a response is equivalent to a Value of 0 except in ranked order contests, where the behavior is implementation-defined.

In a 1-of-M or N-of-M Contest, a VoterInput with Value = 1 would exist for the single Choice or for each of the Choices for which the voter voted.

In a Cumulative Contest, Value may take on values greater than 1. For example, if a voter cast two votes for the first Choice and one vote for the second, a VoterInput with Value = 2 would exist for the first Choice and a VoterInput with Value = 1 would exist for the second Choice.

In a Ranked order Contest, Value contains the ranking that the voter assigns to each Choice.

3.4.5 Constraints

I. For N-of-M and straight party selection Contests, the Value attribute of VoterInput or Endorsement must be 1. For cumulative Contests, $1 \leq$ Value $\leq$ N. (Deliberately, there is no analogous constraint for ranked order Contests.)

II. $N > 0$.

III. $0 \leq$ MaxWriteIns $\leq$ N.

IV. In Contests with CountingLogic = Straight party selection, N = 1 and MaxWriteIns = 0.

V. Every Ballot must be associated with at least one ReportingContext either directly or through its BallotStyle. (Otherwise the Ballot would never be reported.)

VI. A Ballot cannot have a VoterInput for a Choice in a Contest that does not appear in its BallotStyle.

VII. A given BallotStyle may contain at most one Contest with CountingLogic = Straight party selection.

VIII. A Contest with CountingLogic = Straight party selection cannot be straight-party-votable (i.e., there can be no Endorsements referring to its Choices).

IX. In Contests with CountingLogic = Straight party selection, the Names of the Choices must match the Names of Parties.

X. Party names must be unique.

XI. A Ballot may not simultaneously have VoterInput for a Choice and an Alias of that Choice. (The handling of double votes for a given candidate resulting from write-in reconciliation is deliberately unspecified in the VVSG, so for testing purposes it is considered an error.)

XII. A Ballot may not simultaneously have VoterInput in a straight-party-votable Contest and a straight party vote that implies votes in that same Contest. (Resolution of straight party overrides is deliberately unspecified in the VVSG, so for testing purposes they are considered to be errors.)

XIII. The Choice that an Alias cites as canonical cannot be aliased. (Corollary: There can be no cycles or self-referential Aliases.)

XIV. The Choice that an Alias cites as canonical must be in the same Contest.

XV. The Choice referenced by an Endorsement must be canonical (it cannot be an Alias).

XVI. A Ballot cannot have VoterInput for more write-in Choices in a given Contest than is allowed by the MaxWriteIns attribute of the Contest.

3.4.6 Usage for all standard voting variations

3.4.6.1 In-person voting

No special requirements.

3.4.6.2 Absentee voting

Absentee voting is implemented in several different ways in practice, and it can be implemented in several different ways using this model.

1. Absentee Ballots can be tagged with the Absentee category and otherwise mingled with other Ballots.
2. A separate ReportingContext can be created for absentee Ballots and applied to the individual absentee Ballots.
3. A separate BallotStyle can be used for absentee Ballots.

While the first option is the least invasive, absentee Ballots are in practice sometimes processed as a separate precinct, which usually means both a separate ReportingContext and a separate BallotStyle.

3.4.6.3 Review-required ballots

Use Categories and Accepted attributes of Ballot as needed.

3.4.6.4 Write-ins

The number of write-ins permitted is an attribute of the Contest. If the write-in is new, a new Choice is created for it (with IsWriteIn = true). Votes are then associated with that Choice. Alias associations are created as applicable during write-in reconciliation.

3.4.6.5 Split precincts

Ballots are associated with the ReportingContexts pertaining to the applicable Precinct and ElectionDistrict. If different BallotStyles are used for each split, the associations can be made on the BallotStyles. Otherwise, each Ballot must be individually associated.

3.4.6.6 Straight party voting

A single Contest is created with CountingLogic = Straight party selection and Choice Names being equal to the Names of the available Parties. In every other Contest that is straight-party-votable, the straight party behaviors are configured by creating Endorsement associations between the Choices and the Parties.

3.4.6.7 Cross-party endorsement

See straight party voting. Create additional Endorsement associations as needed for multiply endorsed Choices.

3.4.6.8 Ballot rotation

Rotate is a Boolean attribute of Contest. The implementation of variable mapping between Choices and ballot positions is out of scope because ballot positions are abstracted out of the model. However, in paper-based systems, rotation may involve a proliferation of BallotStyles that would have to be added.

3.4.6.9 Primary elections

Create BallotStyles and ReportingContexts as needed to support the different political parties and unaffiliated voters. Non-party-specific Contests appear in all BallotStyles while party-specific Contests only appear in those BallotStyles applicable to the relevant Party.

3.4.6.10 Closed primaries

Assignment of BallotStyles to voters is procedural and out of scope.

3.4.6.11 Open primaries

Assignment of BallotStyles to voters is procedural and out of scope.

3.4.6.12 Provisional / challenged ballots

Use Categories and Accepted attributes of Ballot as needed.

3.4.6.13 1-of-M voting

Set ContestCountingLogic = N-of-M and set N = 1.

3.4.6.14 N-of-M voting

Set ContestCountingLogic = N-of-M and set N appropriately.

3.4.6.15 Cumulative voting

Set ContestCountingLogic = Cumulative and set N appropriately.

3.4.6.16 Ranked order voting

Set ContestCountingLogic = Ranked order and set N appropriately. VoterInput Values specify the rankings as provided on each Ballot.

3.5 Basic schema

The following transforms were used to render the UML model as SQL.

1. At the most basic level, a table represents a class, the columns of the table represent the attributes of that class, and the rows of the table represent the instances of that class.
2. Object identity (*haecceity*) is implemented either by using an existing identifier as primary key or by using a synthetic identifier of integer type, as convenient.
3. Associations to at most 1 instance of another class are implemented using foreign keys within the relevant table with a not-null constraint if the minimum multiplicity is 1. Associations of higher multiplicity are reified as separate tables.
4. Attributes of multiplicity greater than 1 are treated as associations and reified as separate tables.
5. Enums are implemented using the names of the enum values as identifiers. Integrity is maintained by creating a table containing the enum values and making attributes of that enum type into foreign keys on that table.

The classes Tabulator, Precinct and ElectionDistrict are not represented. They were modelled only to clarify how the more general concept ReportingContext relates to the real world and are not needed by the test suite.

```
-- enum
create table BallotCategory (
  Name  Text  primary key
);
insert into BallotCategory values
  ('Early'), ('Regular'), ('InPerson'), ('Absentee'), ('Provisional'),
  ('Challenged'), ('NotRegistered'), ('WrongPrecinct'), ('IneligibleVoter');


-- enum
create table ContestCountingLogic (
  Name  Text  primary key
);
insert into ContestCountingLogic values
  ('N-of-M'), ('Cumulative'), ('Ranked order'), ('Straight party selection');
```

```
-- class
create table ReportingContext (
  Name  Text  primary key
);


-- class
create table Party (
  Name  Text  primary key
);


-- class
create table Contest (
  ContestId      Integer  primary key,
  Description    Text     not null,
  CountingLogic  Text     not null references ContestCountingLogic,
  N              Integer  not null check (N > 0),
  MaxWriteIns    Integer  not null check (MaxWriteIns between 0 and N),
  Rotate         Boolean  not null,

  -- Straight party selections must be 1-of-M with no write-ins.
  check (CountingLogic <> 'Straight party selection' or
          (N = 1 and MaxWriteIns = 0))
);


-- class
create table Choice (
  ChoiceId     Integer  primary key,
  ContestId    Integer  not null references Contest,
  Name         Text     not null,
  Affiliation  Text     references Party,    -- named association
  IsWriteIn    Boolean  not null
);


-- class
create table BallotStyle (
  StyleId  Integer  primary key,
  Name     Text     not null
);


-- class
create table Ballot (
```

```
  BallotId  Integer  primary key,
  StyleId   Integer  not null references BallotStyle,
  Accepted  Boolean  not null
);


-- attribute Ballot::Categories
create table BallotCategoryAssociation (
  BallotId  Integer  references Ballot,
  Category  Text     references BallotCategory,
  primary key (BallotId, Category)
);


-- association class
create table VoterInput (
  BallotId  Integer  references Ballot,
  ChoiceId  Integer  references Choice,
  Value     Integer  not null check (Value > 0),
  primary key (BallotId, ChoiceId)
);


-- association class
create table Endorsement (
  Party     Text     references Party,
  ChoiceId  Integer  references Choice,
  Value     Integer  not null check (Value > 0),
  primary key (Party, ChoiceId)
);


-- named association
create table Alias (
  AliasId   Integer  primary key references Choice,  -- The unwanted alias
  ChoiceId  Integer  not null references Choice,     -- The canonical choice
  check (ChoiceId <> AliasId)                 -- Circular aliases are no good
);


-- unnamed association
create table BallotStyleContestAssociation (
  StyleId    Integer  references BallotStyle,
  ContestId  Integer  references Contest,
  primary key (StyleId, ContestId)
);
```

```
-- unnamed association
create table BallotStyleReportingContextAssociation (
  StyleId           Integer  references BallotStyle,
  ReportingContext  Text     references ReportingContext,
  primary key (StyleId, ReportingContext)
);


-- unnamed association
create table BallotReportingContextAssociation (
  BallotId          Integer  references Ballot,
  ReportingContext  Text     references ReportingContext,
  primary key (BallotId, ReportingContext)
);
```

3.6 Basic test suite

The basic test suite contains 92 tests that exercise different voting variations in small, simple scenarios to isolate the conditions under which failures occur. The basic test suite is intended to be used in conjunction with a volume test that exercises all features together in a large, complex scenario where a significant volume of ballots is processed.

The variations identified in the test suite as AbsenteeVoting and AbsenteeByCategories both conform to the VVSG definition of absentee voting. Tests tagged as AbsenteeByCategories require support for a capability that might not be present in all systems, while AbsenteeVoting tests use a more procedural approach to achieve the same goal. The documented assumption attached to test case 1-basic-AbsenteeByCategory.sql and other tests using the absentee-ballots-by-categories approach means that it is not applicable to absentee voting systems that are limited to the procedural approach. See Section 3.4.6.2 for further discussion on this issue.

Table 4 provides the full list of tests, which breaks down as follows:

- 3 baseline tests that require support for no optional voting variations.
- 19 single-variation tests covering the 12 optional voting variations identified in the basic test suite.
- 66 two-variation tests covering 63 combinations of two voting variations. The other 3 combinations are not meaningful:
  1. AbsenteeByCategories plus AbsenteeVoting: Support for AbsenteeByCategories implies support for AbsenteeVoting.
  2. CrossPartyEndorsement plus StraightPartyVoting: Support for CrossPartyEndorsement implies support for StraightPartyVoting.
  3. CrossPartyEndorsement plus RankedOrderVoting: In ranked order contests, a straight party vote manifests as a particular ranking of choices that is specified by the party. Since all choices must be ranked in any event, there is no additional cross-party functionality to test.
- 1 three-variation test.

- 3 tests that use ballot configurations based on actual sample ballots contributed to the test effort [6]. These tests do not attempt to replicate the reporting structures of the relevant jurisdictions, which include several additional levels of districting and many more ballot configurations.

Several voting variations do not have designated tests:

- *Closed primaries* and *Open primaries* are special cases of *Primary elections*. They are distinguished only by the procedure for assigning BallotStyles to voters, which is beyond the scope of the test suite. They would be tested using the same scenarios as *Primary elections*, but additional requirements on the behavior of the system would apply.

Table 4: Basic test suite

Test case ID / file name	Applies to	Description
1-basic-1ofM.sql	*Voting system*	Small 1-of-M contest, no write-ins, no rejected ballots.
1-basic-AbsenteeByCategory.sql	*Absentee voting*[2]	Small 1-of-M contest with absentee ballots via categories.
1-basic-AbsenteeByCategory-CrossPartyEndorsement.sql	*Absentee voting*[2] ∧ *Cross-party endorsement*	Small straight party + 1-of-M contest with cross-party endorsement, three absentee ballots via categories.
1-basic-AbsenteeByCategory-Cumulative.sql	*Absentee voting*[2] ∧ *Cumulative voting*	Small Cumulative contest, N=3, no write-ins, no rejected ballots, three absentee ballots via categories.
1-basic-AbsenteeByCategory-NofM.sql	*Absentee voting*[2] ∧ *N of M voting*	Small 2-of-M contest, no write-ins, no rejected ballots, three absentee ballots via categories.
1-basic-AbsenteeByCategory-Provisional.sql	*Absentee voting*[2] ∧ *Provisional / challenged ballots*	Small 1-of-M contest with absentee ballots via categories, accepted and rejected provisionals.
1-basic-AbsenteeByCategory-RankedOrder.sql	*Absentee voting*[2] ∧ *Ranked order voting*	Small ranked order contest with absentee ballots via categories.
1-basic-AbsenteeByCategory-StraightParty.sql	*Absentee voting*[2] ∧ *Straight party voting*	Small straight party + 1-of-M contest, absentee ballots via categories.
1-basic-AbsenteeByCategory-WriteIns.sql	*Absentee voting*[2] ∧ *Write-ins*	Small 1-of-M contest with absentee ballots via categories and write-ins, no aliasing.
1-basic-AbsenteeBySpecialPrecinct.sql	*Absentee voting*	Small 1-of-M contest with absentee ballots via a special precinct and ballot style.
1-basic-AbsenteeBySpecialPrecinct-CrossPartyEndorsement.sql	*Absentee voting* ∧ *Cross-party endorsement*	Small straight party + 1-of-M contest with cross-party endorsement, absentee ballots via a special precinct and ballot style.

[2]Assumption: System supports categorization of Ballots. This test is not applicable to systems that require the creation of distinct ballot configurations to implement absentee voting.

1-basic-AbsenteeBySpecialPrecinct-Cumulative.sql	*Absentee voting* ∧ *Cumulative voting*	Small Cumulative contest, N=3, with absentee ballots via a special precinct and ballot style, no write-in, no rejected ballots.
1-basic-AbsenteeBySpecialPrecinct-NofM.sql	*Absentee voting* ∧ *N of M voting*	Small 3-of-M contest with absentee ballots via a special precinct and ballot style, no write-in, no rejected ballots.
1-basic-AbsenteeBySpecialPrecinct-NofM-Provisional.sql	*Absentee voting* ∧ *N of M voting* ∧ *Provisional / challenged ballots*	Small 3-of-M contest with absentee ballots via a special precinct, accepted and rejected provisionals, no write-ins.
1-basic-AbsenteeBySpecialPrecinct-Provisional.sql	*Absentee voting* ∧ *Provisional / challenged ballots*	Small 1-of-M contest with absentee ballots via a special precinct and ballot style, accepted and rejected provisionals.
1-basic-AbsenteeBySpecialPrecinct-RankedOrder.sql	*Absentee voting* ∧ *Ranked order voting*	Small ranked order contest with absentee ballots via a special precinct and ballot style.
1-basic-AbsenteeBySpecialPrecinct-SplitPrecinct.sql	*Absentee voting* ∧ *Split precincts*	Small 1-of-M contest with absentee ballots via a special precinct and a split precinct.
1-basic-AbsenteeBySpecialPrecinct-StraightParty.sql	*Absentee voting* ∧ *Straight party voting*	Small straight party + 1-of-M contest, absentee ballots via a special precinct and ballot style.
1-basic-AbsenteeBySpecialPrecinct-WriteIns.sql	*Absentee voting* ∧ *Write-ins*	Small 1-of-M contest with absentee ballots via a special precinct and write-ins.
1-basic-AbsenteeBySpecialPrecinct-Yes-or-No.sql	*Absentee voting*	Small Yes-or-No contest with absentee ballots via a special precinct and ballot style.
1-basic-BallotRotation.sql	*Ballot rotation*	Small 1-of-M contest with ballot rotation.
1-basic-BallotRotation-AbsenteeByCategory.sql	*Ballot rotation* ∧ *Absentee voting*[2]	Small 1-of-M contest with ballot rotation and absentee ballots via categories.
1-basic-BallotRotation-AbsenteeBySpecialPrecinct.sql	*Ballot rotation* ∧ *Absentee voting*	Small 1-of-M contest with ballot rotation, absentee ballots via a special precinct.
1-basic-BallotRotation-CrossPartyEndorsement.sql	*Ballot rotation* ∧ *Cross-party endorsement*	Small straight party + 1-of-M contest with cross-party endorsement and ballot rotation.

1-basic-BallotRotation-Cumulative.sql	*Ballot rotation* ∧ *Cumulative voting*	Small cumulative voting contest, N=2, with ballot rotation, no write-ins, no rejected ballots.
1-basic-BallotRotation-NofM.sql	*Ballot rotation* ∧ *N of M voting*	Small 2-of-M contest, with ballot rotation.
1-basic-BallotRotation-Provisional.sql	*Ballot rotation* ∧ *Provisional / challenged ballots*	Small 1-of-M contest with ballot rotation, accepted and rejected provisionals.
1-basic-BallotRotation-RankedOrder.sql	*Ballot rotation* ∧ *Ranked order voting*	Small ranked order contest with ballot rotation.
1-basic-BallotRotation-StraightParty.sql	*Ballot rotation* ∧ *Straight party voting*	Small straight party + 1-of-M contest with ballot rotation.
1-basic-BallotRotation-WriteInsAliases.sql	*Ballot rotation* ∧ *Write-ins*	Small 1-of-M contest with write-ins and aliases and ballot rotation.
1-basic-CrossPartyEndorsement.sql	*Cross-party endorsement*	Small straight party + 1-of-M contest with cross-party endorsement.
1-basic-CrossPartyEndorsement-Cumulative.sql	*Cross-party endorsement* ∧ *Cumulative voting*	Small straight party + cumulative contest with cross-party endorsement, no write-ins, no rejected ballots.
1-basic-CrossPartyEndorsement-NofM.sql	*Cross-party endorsement* ∧ *N of M voting*	Small straight party + 2-of-M contest with cross-party endorsement.
1-basic-CrossPartyEndorsement-Provisional.sql	*Cross-party endorsement* ∧ *Provisional / challenged ballots*	Small straight party + 1-of-M contest with cross-party endorsement, accepted and rejected provisionals.
1-basic-CrossPartyEndorsement-WriteIns.sql	*Cross-party endorsement* ∧ *Write-ins*	Small straight party + 1-of-M contest with cross-party endorsement and write-ins.
1-basic-Cumulative.sql	*Cumulative voting*	Small cumulative voting contest, no write-ins, no rejected ballots.
1-basic-Cumulative-NofM.sql	*Cumulative voting* ∧ *N of M voting*	Small cumulative voting contest plus small N-of-M contest, no write-ins, no rejected ballots.
1-basic-Cumulative-Provisional.sql	*Cumulative voting* ∧ *Provisional / challenged ballots*	Small cumulative voting contest with accepted and rejected provisionals.
1-basic-Cumulative-RankedOrder.sql	*Cumulative voting* ∧ *Ranked order voting*	Small ranked order contest plus small cumulative contest, no special cases.

1-basic-NoBallots-1ofM.sql	*Voting system*	Small 1-of-M contest with no ballots cast.
1-basic-NoBallots-RankedOrder.sql	*Ranked order voting*	Small ranked order contest with no ballots cast.
1-basic-NoChoices-1ofM.sql	*Write-ins*	Small 1-of-M contest with no choices (write-ins only).
1-basic-NoChoicesNoBallots-1ofM.sql	*Write-ins*	Small 1-of-M contest with no choices (write-ins only) and no ballots.
1-basic-NofM.sql	*N of M voting*	Small 2-of-M contest, no write-ins, no rejected ballots.
1-basic-NofM-Provisional.sql	*N of M voting* $\wedge$ *Provisional / challenged ballots*	Small 2-of-M contest with accepted and rejected provisionals.
1-basic-NofM-WriteIns.sql	*N of M voting* $\wedge$ *Write-ins*	Small 2-of-M contest with write-ins, no aliasing.
1-basic-Primary.sql	*Primary elections*	Small primary election, no write-ins, no rejected ballots.
1-basic-Primary-AbsenteeByCategory.sql	*Primary elections* $\wedge$ *Absentee voting*[2]	Small primary election with absentee ballots via categories, no write-ins, no rejected ballots.
1-basic-Primary-AbsenteeBySpecialPrecinct.sql	*Primary elections* $\wedge$ *Absentee voting*	Small primary election with absentee ballots via a special precinct and ballot style, no write-ins, no rejected ballots.
1-basic-Primary-BallotRotation.sql	*Primary elections* $\wedge$ *Ballot rotation*	Small primary election with ballot rotations, no write-ins, no rejected ballots.
1-basic-Primary-CrossPartyEndorsement.sql	*Primary elections* $\wedge$ *Cross-party endorsement*	Small primary election with straight party voting and cross-party endorsement in a non-party-specific contest.
1-basic-Primary-Cumulative.sql	*Primary elections* $\wedge$ *Cumulative voting*	Small primary election with cumulative voting, no write-ins, no rejected ballots.
1-basic-Primary-NofM.sql	*Primary elections* $\wedge$ *N of M voting*	Small primary election with N-of-M voting, no write-ins, no rejected ballots.
1-basic-Primary-Provisional.sql	*Primary elections* $\wedge$ *Provisional / challenged ballots*	Small primary election, no write-ins, with accepted and rejected provisionals.

1-basic-Primary-RankedOrder.sql	*Primary elections* ∧ *Ranked order voting*	Small primary election with ranked order voting, no write-ins, no rejected ballots.
1-basic-Primary-SplitPrecinct.sql	*Primary elections* ∧ *Split precincts*	Small primary election with a split precinct, no write-ins, no rejected ballots.
1-basic-Primary-StraightParty.sql	*Primary elections* ∧ *Straight party voting*	Small primary election with straight party voting in a non-party-specific contest.
1-basic-Primary-WriteIns.sql	*Primary elections* ∧ *Write-ins*	Small primary election with write-ins, no rejected ballots.
1-basic-Provisional.sql	*Provisional / challenged ballots*	Small 1-of-M contest with accepted and rejected provisionals.
1-basic-RankedOrder-1.sql	*Ranked order voting*	Small ranked order contest, N=1, M=4, no special cases.
1-basic-RankedOrder-2.sql	*Ranked order voting*	Small ranked order contest, N=2, M=4, no special cases.
1-basic-RankedOrder-NofM.sql	*Ranked order voting* ∧ *N of M voting*	Small ranked order contest plus small N-of-M contest, no special cases.
1-basic-RankedOrder-Provisional.sql	*Ranked order voting* ∧ *Provisional / challenged ballots*	Small ranked order contest, N=1, M=4, with accepted and rejected provisionals.
1-basic-RankedOrder-WriteIns.sql	*Ranked order voting* ∧ *Write-ins*	Small ranked order contest, N=1, M=4, with write-in.
1-basic-SplitPrecinct-1.sql	*Split precincts*	Small 1-of-M contest with a split precinct.
1-basic-SplitPrecinct-2.sql	*Split precincts*	Two districts, three precincts (one split), three contests, four ballot styles, forty ballots.
1-basic-SplitPrecinct-AbsenteeByCategory.sql	*Split precincts* ∧ *Absentee voting*[2]	Small 1-of-M contest with a split precinct and absentee ballots via categories.
1-basic-SplitPrecinct-BallotRotation.sql	*Split precincts* ∧ *Ballot rotation*	Like SplitPrecinct-2 except with ballot rotation.
1-basic-SplitPrecinct-CrossPartyEndorsement.sql	*Split precincts* ∧ *Cross-party endorsement*	Like SplitPrecinct-2 except with straight party voting and cross-party endorsement.
1-basic-SplitPrecinct-Cumulative.sql	*Split precincts* ∧ *Cumulative voting*	Small Cumulative contest, N=2, with a split precinct.

1-basic-SplitPrecinct-NofM.sql	*Split precincts* ∧ *N of M voting*	Small 3-of-M contest with a split precinct, no write-ins, no rejected ballots.
1-basic-SplitPrecinct-Provisional.sql	*Split precincts* ∧ *Provisional / challenged ballots*	Small 1-of-M contest with a split precinct and accepted and rejected provisionals.
1-basic-SplitPrecinct-RankedOrder.sql	*Split precincts* ∧ *Ranked order voting*	Like SplitPrecinct-2 except with ranked order contests at the district level.
1-basic-SplitPrecinct-StraightParty.sql	*Split precincts* ∧ *Straight party voting*	Like SplitPrecinct-2 except with straight party voting.
1-basic-SplitPrecinct-WriteIns.sql	*Split precincts* ∧ *Write-ins*	Small 1-of-M contest with split precinct and write-in.
1-basic-StraightParty.sql	*Straight party voting*	Small straight party + 1-of-M contest, no write-ins, no rejected ballots.
1-basic-StraightParty-Cumulative.sql	*Straight party voting* ∧ *Cumulative voting*	Small straight party + cumulative contest, no write-ins, no rejected ballots.
1-basic-StraightParty-NofM.sql	*Straight party voting* ∧ *N of M voting*	Small straight party + 2-of-M contest, no write-ins, no rejected ballots.
1-basic-StraightParty-Provisional.sql	*Straight party voting* ∧ *Provisional / challenged ballots*	Small straight party + 1-of-M contest with accepted and rejected provisionals.
1-basic-StraightParty-RankedOrder.sql	*Straight party voting* ∧ *Ranked order voting*	Small ranked order contest, N=1, M=4, with straight-party voting.
1-basic-StraightParty-WriteIns.sql	*Straight party voting* ∧ *Write-ins*	Small straight party + 1-of-M contest with write-ins.
1-basic-WriteIns.sql	*Write-ins*	Small 1-of-M contest with write-ins, no aliasing.
1-basic-WriteInsAliases.sql	*Write-ins*	Small 1-of-M contest with write-ins and aliases.
1-basic-WriteInsAliases-AbsenteeByCategory.sql	*Write-ins* ∧ *Absentee voting*[2]	Small 1-of-M contest with write-ins, aliases and absentee ballots via categories.
1-basic-WriteInsAliases-AbsenteeBySpecialPrecinct.sql	*Write-ins* ∧ *Absentee voting*	Small 1-of-M contest with write-ins, aliases, absentee ballots via a special precinct and ballot style.
1-basic-WriteInsAliases-Cumulative.sql	*Write-ins* ∧ *Cumulative voting*	Small Cumulative contest, N=2, with write-ins and aliases, no rejected ballots.

1-basic-WriteInsAliases-NofM.sql	*Write-ins* $\wedge$ *N of M voting*	Small 2-of-M contest with write-ins and aliases, no rejected ballots.
1-basic-WriteInsAliases-Provisional.sql	*Write-ins* $\wedge$ *Provisional / challenged ballots*	Small 1-of-M contest with write-ins, aliases, accepted and rejected provisionals.
1-basic-Yes-or-No.sql	*Voting system*	Small Yes-or-No contest, no rejected ballots.
1-basic-samples-AlleganyGeneral2004.sql	*Write-ins* $\wedge$ *N of M voting*	Test spec based on Allegany County, MD sample ballot for congressional district 6, general election, 2004-11-02.
1-basic-samples-CecilRPrimary1998.sql	*N of M voting*	Test spec based on Cecil County, MD sample ballot for Republican primary election, 1st congressional district, legislative district 35, 1998-09-15. The generated test does not include anything specific to primary elections.
1-basic-samples-FairfaxGeneral2004.sql	*Write-ins*	Test spec based on Fairfax County, VA sample ballot for 8th district, general election, 2004-11-02.

3.7 Required test cases not included in the basic Votetest test suite

Some requirements that might logically be tested during the phase when Votetest is used do not have associated test cases because the behaviors in question are orthogonal to the Votetest data model and/or so dependent on the specifics of the implementation that only an abstract test script could be provided. Those requirements and descriptions of the needed test cases are shown in Table 5.

Table 5: Additional required test cases

Requirements	Test description
Realistic ballot styles [1, Volume I Req. 2.2.1.1.a, 2.2.1.2.c and 4.1.3.1.d]	Test a variety of sample ballot styles, such as those available on the NIST web site [6], to ensure that the voting system can handle the text of long ballot questions, complicated choice labels (such as for primary presidential delegation nominations), miscellaneous ballot text, and overall formatting and layout of ballot styles as used in practice.
Optical scan accuracy [1, Volume I Sections 4.1.1, 4.1.5.2, and 4.1.6.1]	When test cases are realized for optical scan devices, the marks used should exercise the range of reliably detectable marks as defined by the vendor. However, to ensure that the tests are defensible, the relative frequency with which the different types of marks appear should be realistic. While both ideal marks and poor marks should be tested repeatedly, it would not be realistic for either extreme to dominate the input. The average mark should be typical of what an average voter would make. The ability of the scanner to ignore extraneous marks should also be tested ([1, Volume I Req. 4.1.5.2.e]).
Marginal marks [1, Volume I Req. 3.2.2.2-E]	Test marks that are clearly within the marginal zone as defined by the vendor to verify that the behavior on marginal marks is as specified. Do not test marks that are near the boundaries, as the uncertainty of the boundaries is of no consequence.

Respecting limits [1, Volume I Req. 1.5.2.a.iv and Volume II Section A.3.5]	Construct test cases as needed to verify that the system and its constituent devices are able to operate correctly at the documented limits. If an implementation limit is sufficiently great that it cannot be verified through operational testing without severe expense and hardship, the test lab shall attest this in the test report and substitute a combination of design review, logic verification, and operational testing to a reduced limit. The test generator (Section 5.7) may be of use.
Definition reuse [1, Volume I Req. 2.2.1.2.e and g]	Choose two test cases that use similar ballot styles. Execute the first as usual, but instead of clearing out all definitions between tests, attempt to construct the ballot styles for the second test based on the definitions retained from the first.
Validation of input [1, Volume I Req. 2.1.5.1.b.vi]	Since poor validation of input often creates an attack vector, this should be covered as part of security testing and evaluation.
Miscellaneous capabilities [1, Volume I Req. 2.1.4.j, 2.1.7.1.a, 2.4.3.e, 4.1.5.1.c and many others]	The VVSG contains numerous requirements for miscellaneous voting system capabilities that are orthogonal to the Votetest data model. Since the general test template specified in Section 3.3.4.2 treats each test case as an entire election, many such capabilities will be tested incidentally during the execution of the basic test suite. Use the Voting Equipment User Documentation to determine how to exercise any capabilities that are not tested incidentally and then verify that they satisfy the applicable VVSG requirements.
System-specific functions	These must be tested in accordance with [1, Volume II Sections 3.2.3 and 6.7].

3.8 Requirements trace

Table 6 clarifies the traceability of tests to each requirement in Volume I Chapter 2 of the VVSG [1].

Table 6: Traceability to Volume I Chapter 2 requirements

Section 2.1.1	Out of Votetest scope (security).
Req. 2.1.2.h	All Votetest tests are traceable to this requirement.
Req. 2.1.2.i	All Votetest tests are traceable to this requirement.
Req. 2.1.2.j	All Votetest tests are traceable to this requirement.

Req. 2.1.2.k	Design requirement, tested by inspection.
Req. 2.1.2.l	Design requirement, tested by inspection.
Req. 2.1.2.a	Design requirement, tested by inspection.
Section 2.1.3	Requires design review; applies whenever failures occur.
Req. 2.1.4.a	Requires design review; operationally tested only by exception.
Req. 2.1.4.b	Out of Votetest scope (hardware test).
Req. 2.1.4.c	Out of Votetest scope (hardware test).
Req. 2.1.4.d	Out of Votetest scope (hardware test).
Req. 2.1.4.e	Requires design review; operationally tested only by exception.
Req. 2.1.4.f	Out of Votetest scope (security).
Req. 2.1.4.g	All Votetest tests are traceable to this requirement.
Req. 2.1.4.h	Out of Votetest scope (security).
Req. 2.1.4.i	Vague requirement, applicable in exceptional cases.
Req. 2.1.4.j	See Section 3.7 (miscellaneous capabilities).
Req. 2.1.4.k	Requires design review; operationally tested only by exception.
Req. 2.1.4.l	See Section 3.7 (miscellaneous capabilities).
Req. 2.1.5.1.a (i–vii)	Out of Votetest scope (security).
Req. 2.1.5.1.b.i	Requires design review; applies whenever errors occur.
Req. 2.1.5.1.b.ii	Out of Votetest scope (usability).
Req. 2.1.5.1.b.iii	Design requirement, tested by inspection.
Req. 2.1.5.1.b.iv	Out of Votetest scope (usability).
Req. 2.1.5.1.b.v	Out of Votetest scope (usability).
Req. 2.1.5.1.b.vi	Requires design review; applies whenever erroneous responses occur; also overlaps security (see Section 3.7).
Req. 2.1.5.1.b.vii	Requires design review; applies whenever nested failures occur.
Req. 2.1.5.1.c ¶1	Vague requirement, applicable in exceptional cases.
Req. 2.1.5.1.c ¶2	Out of Votetest scope (usability).
Req. 2.1.5.1.c ¶3	See Section 3.7 (miscellaneous capabilities).
Section 2.1.5.2	Out of Votetest scope (security).
Section 2.1.6	All Votetest tests are traceable to this requirement.
Req. 2.1.7.1.a	See Section 3.7 (miscellaneous capabilities).
Req. 2.1.7.1.b	All Votetest tests are traceable to this requirement.
Req. 2.1.7.1.c	All Votetest tests are traceable to this requirement.
Req. 2.1.7.1.d	All tests except 1-basic-1ofM.sql, 1-basic-NoBallots-1ofM.sql and 1-basic-Yes-or-No.sql are traceable to this requirement.
Section 2.1.7.2	Documentation requirement, tested by inspection.
Req. 2.1.8.a	All Votetest tests are traceable to this requirement.
Req. 2.1.8.b	All Votetest tests are traceable to this requirement.
Req. 2.1.8.c	Requires design review; overlaps security.
Req. 2.1.8.d	Out of Votetest scope (security).
Req. 2.1.8.e	Design requirement, tested by inspection.
Section 2.1.9	Out of Votetest scope (security).
Section 2.1.10	Out of Votetest scope (procedural requirement).
Req. 2.2.1.1.a	All Votetest tests are traceable to this requirement; see also Section 3.7 regarding testing of realistic ballot styles.
Req. 2.2.1.1.b (i–iii)	All Votetest tests are traceable to this requirement.
Req. 2.2.1.1.c	Out of Votetest scope (capacity test).
Req. 2.2.1.1.d	All PrimaryElections tests are traceable to this requirement.

Req. 2.2.1.1.e	See Section 3.7 (miscellaneous capabilities).
Req. 2.2.1.1.f	All Votetest tests are traceable to this requirement.
Req. 2.2.1.1.a	Vague requirement, applicable in exceptional cases.
Req. 2.2.1.1.b	All Votetest tests are traceable to this requirement.
Req. 2.2.1.2.a	All Votetest tests are traceable to this requirement.
Req. 2.2.1.2.b	Out of Votetest scope (usability).
Req. 2.2.1.2.c	Vague requirement, applicable in exceptional cases; see also Section 3.7 regarding testing of realistic ballot styles.
Req. 2.2.1.2.d	Out of Votetest scope (capacity test).
Req. 2.2.1.2.e	See Section 3.7 (definition reuse).
Req. 2.2.1.2.f	Out of Votetest scope (security).
Req. 2.2.1.2.g	See Section 3.7 (definition reuse).
Req. 2.2.1.3.a	Out of Votetest scope (usability).
Req. 2.2.1.3.b	All Votetest tests are traceable to this requirement.
Req. 2.2.1.3.c	Design requirement, tested by inspection.
Section 2.2.1.3, final ¶	Documentation requirement, tested by inspection.
Req. 2.2.2.a	All Votetest tests are traceable to this requirement.
Req. 2.2.2.b	All Votetest tests are traceable to this requirement.
Req. 2.2.2.c	All AbsenteeBySpecialPrecinct tests, all Primary tests, all SplitPrecinct tests, and all samples tests are traceable to this requirement.
Req. 2.2.2.d	Untestable requirement (depends on jurisdiction law).
Req. 2.2.2.e	All Votetest tests are traceable to this requirement.
Req. 2.2.3.a	Documentation requirement, tested by inspection.
Req. 2.2.3.b	All Votetest tests are traceable to this requirement.
Req. 2.2.3.c	All Votetest tests are traceable to this requirement.
Req. 2.2.4.d	All Votetest tests are traceable to this requirement.
Req. 2.2.4.e	All Votetest tests are traceable to this requirement.
Req. 2.2.4.f	All Votetest tests are traceable to this requirement.
Req. 2.2.4.g	All Votetest tests are traceable to this requirement.
Req. 2.2.4.h	All Votetest tests are traceable to this requirement.
Req. 2.2.4.i	Requires design review; operationally tested only by exception.
Req. 2.2.4.j	Requires design review; operationally tested only by exception.
Req. 2.2.4.k	Requires design review; operationally tested only by exception.
Req. 2.2.4.l	All Votetest tests are traceable to this requirement.
Req. 2.2.4.m	All Votetest tests are traceable to this requirement.
Section 2.2.5, a–i	All Votetest tests are traceable to this requirement.
Section 2.2.5, final ¶	All Votetest tests are traceable to this requirement.
Req. 2.2.6.a	All Votetest tests are traceable to this requirement.
Req. 2.2.6.b	All Votetest tests are traceable to this requirement.
Req. 2.2.6.c	Vague requirement, applicable in exceptional cases.
Req. 2.3.1.1.a	See Section 3.7 (miscellaneous capabilities).
Req. 2.3.1.1.b	See Section 3.7 (miscellaneous capabilities).
Req. 2.3.1.2.a	See Section 3.7 (miscellaneous capabilities).
Req. 2.3.1.2.b	Design requirement, tested by inspection.
Req. 2.3.1.2.c	Design requirement, tested by inspection.
Req. 2.3.1.2.d	All Votetest tests are traceable to this requirement.
Req. 2.3.1.2.e	All Votetest tests are traceable to this requirement.
Req. 2.3.1.2.f	Requires design review; applies whenever failures occur.

Req. 2.3.1.3.a	Out of Votetest scope (security).
Req. 2.3.1.3.b	See Section 3.7 (miscellaneous capabilities).
Req. 2.3.1.3.c	All Votetest tests are traceable to this requirement.
Req. 2.3.1.3.d	Requires design review; applies whenever failures occur.
Req. 2.3.2.a	All AbsenteeBySpecialPrecinct tests, all Primary tests, all SplitPrecinct tests, and all samples tests are traceable to this requirement.
Req. 2.3.2.b	All Votetest tests are traceable to this requirement.
Req. 2.3.2.c	Out of Votetest scope (security).
Req. 2.3.2.d	Out of Votetest scope (security).
Req. 2.3.2.e	All tests except for Primary tests are traceable to this requirement.
Req. 2.3.2.f	All Primary tests are traceable to this requirement.
Req. 2.3.2.g	All Votetest tests are traceable to this requirement.
Req. 2.3.2.h	All AbsenteeBySpecialPrecinct tests, all Primary tests, all SplitPrecinct tests, and all samples tests are traceable to this requirement.
Req. 2.3.3.1.a	Out of Votetest scope (usability).
Req. 2.3.3.1.b	Out of Votetest scope (security).
Req. 2.3.3.1.c	All Votetest tests are traceable to this requirement.
Req. 2.3.3.1.d	All WriteIns tests are traceable to this requirement.
Req. 2.3.3.1.e	Out of Votetest scope (hardware test).
Req. 2.3.3.1.f	Out of Votetest scope (hardware test).
Req. 2.3.3.2.a	Out of Votetest scope (usability).
Req. 2.3.3.2.b	All Votetest tests are traceable to this requirement.
Req. 2.3.3.2.c	All Votetest tests are traceable to this requirement.
Req. 2.3.3.2.d	Out of Votetest scope (security).
Req. 2.3.3.2.e	All Votetest tests are traceable to this requirement.
Req. 2.3.3.2.f	All Votetest tests are traceable to this requirement.
Req. 2.3.3.2.g	All Votetest tests are traceable to this requirement.
Req. 2.3.3.2.h	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.a	Out of Votetest scope (security).
Req. 2.3.3.3.b	Out of Votetest scope (usability).
Req. 2.3.3.3.c	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.d	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.e	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.f	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.g	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.h	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.i	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.j	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.k	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.l	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.m	Requires design review; applies whenever failures occur.
Req. 2.3.3.3.n	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.o	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.p	Out of Votetest scope (security).
Req. 2.3.3.3.q	See Section 3.7 (miscellaneous capabilities).
Req. 2.3.3.3.r	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.s	Out of Votetest scope (security).
Req. 2.3.3.3.t	Out of Votetest scope (security).

Req. 2.3.3.3.u	All Votetest tests are traceable to this requirement.
Req. 2.3.3.3.v	All Votetest tests are traceable to this requirement.
Req. 2.4.1.a	Out of Votetest scope (security).
Req. 2.4.1.b	See Section 3.7 (miscellaneous capabilities).
Req. 2.4.1.c	Design requirement, tested by inspection.
Req. 2.4.1.d	All Votetest tests are traceable to this requirement.
Req. 2.4.1.e	Out of Votetest scope (security).
Req. 2.4.2	All Votetest tests are traceable to this requirement.
Req. 2.4.3.a	All Votetest tests are traceable to this requirement.
Req. 2.4.3.b	All Votetest tests are traceable to this requirement.
Req. 2.4.3.c	All Votetest tests are traceable to this requirement.
Req. 2.4.3.d	All Votetest tests are traceable to this requirement.
Req. 2.4.3.e	See Section 3.7 (miscellaneous capabilities).
Req. 2.4.3.f	All Votetest tests are traceable to this requirement.
Req. 2.4.3.g	Requires design review; operationally tested only by exception.
Req. 2.4.3.a	Out of Votetest scope (security).
Req. 2.4.3.b	See Section 3.7 (miscellaneous capabilities).
Req. 2.4.3.c	All Votetest tests are traceable to this requirement.
Req. 2.4.3.d	Requires design review; operationally tested only by exception.
Section 2.4.4	Out of Votetest scope (security).
Section 2.4.5 (a–c)	See Section 3.7 (miscellaneous capabilities).
Section 2.5	Out of Votetest scope (hardware test).

4 Advanced schema

The schema for core requirements is designed only for testing. It does not respond to the security, privacy, accessibility, or usability requirements of the VVSG.

The schema is built in five layers:

1. Translation of the data model. This layer contains all of the tables and data. The other layers are comprised entirely of views.
2. Conveniences defined over the data model.
3. Adaptation layer. This layer translates the raw voter inputs per the data model into the effective voter inputs required by the logic model.
4. Data integrity checks.
5. Translation of the logic model.

The schema was developed with PostgreSQL [7] running on a GNU/Linux operating system. It uses extensions to the SQL standard [2] that might not function as intended with other databases.

Layer 1, the translation of the data model, was covered in Section 3.5. The following subsections cover the other four layers.

4.1 Conveniences

All assertions have as a precondition the assumption that all of the constraints are satisfied (see Section 3.4.5).

4.1.1 FilteredBallotContestAssociation

The view FilteredBallotContestAssociation identifies all Contests that appear on a given Ballot, excluding ranked order Contests.

```
create view FilteredBallotContestAssociation (BallotId, ContestId) as
  select BallotId, ContestId
    from Ballot
      natural join BallotStyleContestAssociation
      natural join Contest
    where CountingLogic <> 'Ranked order';
```

Assertion 1 *For each Ballot, FilteredBallotContestAssociation contains exactly one row for each non-ranked-order Contest appearing in the BallotStyle identified by Ballot.StyleId, and zero rows for any ranked order Contests.*

1. For each Ballot in table Ballot, the result of Ballot natural join BallotStyleContestAssociation contains one row for each Contest appearing in the BallotStyle identified by Ballot.StyleId. The primary key constraints on Ballot and BallotStyleContestAssociation ensure that there cannot be more than one such row.

2. The primary key constraint on table Contest and the foreign key constraint on table BallotStyleContestAssociation ensure that the natural join of the previous result with table Contest will not drop rows. Consequently, for each Ballot, there is still one row for each Contest appearing in the BallotStyle identified by Ballot.StyleId.

3. The where clause removes rows pertaining to ranked order Contests.

Assertion 2 *If the BallotStyle contains zero Contests, BallotStyleContestAssociation contains no rows with that BallotId.*

If BallotStyleContestAssociation contains no rows with a StyleId matching Ballot.StyleId, meaning that the BallotStyle contains no Contests, then the natural join of Ballot and BallotStyleContestAssociation produces no rows for that Ballot. The subsequent natural join with Contest cannot add back a BallotId that was dropped, so FilteredBallotContestAssociation correctly contains no rows for that Ballot.

4.1.2 ReportingContextAssociationMerge

The view ReportingContextAssociationMerge merges ReportingContexts inherited from the BallotStyle with ReportingContexts specified on Ballot instances. Duplicates are suppressed.

```
create view ReportingContextAssociationMerge (BallotId, ReportingContext) as
    select BallotId, ReportingContext
      from BallotReportingContextAssociation
  union
    select BallotId, ReportingContext
      from Ballot natural join BallotStyleReportingContextAssociation;
```

Assertion 3 *For each Ballot, ReportingContextAssociationMerge contains exactly one row for each relevant ReportingContext.*

1. The only two ways by which a ReportingContext may be relevant to a Ballot are by association with a Ballot's BallotStyle or by association with the Ballot itself.

2. If a ReportingContext is associated with the BallotStyle, a corresponding (BallotId, ReportingContext) tuple will be projected from the result of Ballot natural join BallotStyleReportingContextAssociation.

3. If a ReportingContext is associated with the Ballot itself, a corresponding (BallotId, ReportingContext) tuple will be selected from BallotReportingContextAssociation.

4. Duplicate (BallotId, ReportingContext) tuples are suppressed by the union operator.

Assertion 4 *For each Ballot, ReportingContextAssociationMerge contains at least one row.*

The case where no ReportingContext is relevant to a given Ballot is prohibited by Constraint V and is detected by the integrity view UnreportedBallots (see Section 4.3).

4.1.3 VotableChoices

The view VotableChoices identifies all canonical Choices for which a valid VoterInput could exist (those contained in the applicable BallotStyles), excluding Aliases.

```
create view VotableChoices (BallotId, ChoiceId) as
  select BallotId, ChoiceId
    from Ballot
      natural join BallotStyleContestAssociation
      natural join Choice
    where ChoiceId not in
      (select AliasId from Alias);
```

Assertion 5 *For each Ballot, VotableChoices contains exactly one row for each canonical Choice for which the Ballot could contain a vote.*

1. For each Ballot in table Ballot, the result of Ballot natural join BallotStyleContestAssociation contains one row for each Contest appearing in the BallotStyle identified by Ballot.StyleId. The primary key constraints on Ballot and BallotStyleContestAssociation ensure that there cannot be more than one such row.

2. For each Ballot, joining Choice with the previous result produces exactly one row for each Choice pertaining to any Contest appearing on the Ballot. The primary key constraint on Choice ensures that there cannot be more than one such row.

3. By definition, a canonical Choice does not appear in the AliasId column of table Alias, so the where clause does not eliminate any canonical Choices.

Assertion 6 *VotableChoices contains no rows pertaining to any non-canonical Choices (Aliases).*

The where clause eliminates rows pertaining to non-canonical Choices (those appearing in the AliasId column of table Alias).

Assertion 7 *If the BallotStyle of a given Ballot contains zero Contests, VotableChoices contains no rows with that BallotId.*

If BallotStyleContestAssociation contains no rows with a StyleId matching Ballot.StyleId, meaning that the BallotStyle contains no Contests, then the natural join of Ballot and BallotStyleContestAssociation produces no rows for that Ballot. The subsequent natural join with Choice cannot add back a BallotId that was dropped, so VotableChoices correctly contains no rows for that Ballot.

Assertion 8 *If the BallotStyle of a given Ballot contains only Contests having zero Choices, VotableChoices contains no rows with that BallotId.*

If the BallotStyle of a given Ballot contains only Contests having zero Choices, then the natural join with Choice will eliminate all rows containing that BallotId.

4.1.4 ReportingContextContestAssociation

The view ReportingContextContestAssociation identifies all Contests that are relevant in a given ReportingContext. This includes those appearing in a BallotStyle associated with the ReportingContext and those appearing in a Ballot associated with the ReportingContext. A BallotStyle association can make a Contest relevant even if there are no applicable Ballots.

```
create view ReportingContextContestAssociation (ReportingContext, ContestId) as
    select ReportingContext, ContestId
      from BallotStyleReportingContextAssociation
        natural join BallotStyleContestAssociation
  union
    select ReportingContext, ContestId
      from BallotReportingContextAssociation
        natural join Ballot
        natural join BallotStyleContestAssociation;
```

Assertion 9 *For each ReportingContext, ReportingContextContestAssociation contains exactly one row for each Contest that is relevant in that ReportingContext.*

1. The relevance of a Contest within a ReportingContext happens through an intermediary BallotStyle. The association of Contests with BallotStyles is specified by the table BallotStyleContestAssociation.

2. A BallotStyle can become relevant to a ReportingContext in two ways: directly, via BallotStyleReportingContextAssociation, or indirectly, via BallotReportingContextAssociation and Ballot.StyleId.

3. If a BallotStyle is directly relevant to a ReportingContext, a corresponding (ReportingContext, ContestId) tuple will be projected from the natural join of BallotStyleReportingContextAssociation and BallotStyleContestAssociation.

4. If a BallotStyle is indirectly relevant to a ReportingContext, there must exist a Ballot having that BallotStyle that is associated with that ReportingContext via BallotReportingContextAssociation. BallotReportingContextAssociation natural join Ballot will therefore contain one row identifying that Ballot and that ReportingContext. The subsequent natural join with BallotStyleContestAssociation is on the column StyleId; constraints ensure that this join will not drop rows. Consequently, a (ReportingContext, ContestId) tuple corresponding to the indirect association between the ReportingContext and the Contest will be projected from the three-way join.

5. Duplicate (ReportingContext, ContestId) tuples are suppressed by the union operator.

4.1.5 FilteredContextContestAssociation

The view FilteredContextContestAssociation is the same as ReportingContextContestAssociation except it excludes ranked order Contests.

```
create view FilteredContextContestAssociation (ReportingContext, ContestId) as
  select ReportingContext, ContestId
    from ReportingContextContestAssociation
      natural join Contest
    where CountingLogic <> 'Ranked order';
```

Assertion 10 *For each ReportingContext, FilteredContextContestAssociation contains exactly one row for each non-ranked-order Contest that is relevant in that ReportingContext, and zero rows for each ranked-order Contest.*

Per Assertion 9, for each ReportingContext, ReportingContextContestAssociation contains exactly one row for each Contest that is relevant in that ReportingContext. Constraints ensure that the natural join with Contest will not drop rows. The only difference from ReportingContextContestAssociation therefore is the elimination of ranked order Contests by the where clause.

4.1.6 FilteredContextChoiceAssociation

The view FilteredContextChoiceAssociation identifies all Choices that are relevant in a given ReportingContext, excluding Aliases and Choices from ranked order Contests.

```
create view FilteredContextChoiceAssociation (ReportingContext, ChoiceId) as
  select ReportingContext, ChoiceId
    from FilteredContextContestAssociation
      natural join Choice
    where ChoiceId not in
      (select AliasId from Alias);
```

Assertion 11 *For each ReportingContext, FilteredContextChoiceAssociation contains exactly one row for each canonical Choice in each non-ranked-order Contest that is relevant in that ReportingContext, and zero rows for any other Choice.*

Per Assertion 10, for each ReportingContext, FilteredContextContestAssociation contains exactly one row for each non-ranked-order Contest that is relevant in that ReportingContext. For each ReportingContext, the result of FilteredContextContestAssociation natural join Choice contains exactly one row for each Choice in each non-ranked-order Contest that is relevant in that ReportingContext. The where clause eliminates rows pertaining to non-canonical Choices (Aliases).

4.1.7 BallotCounts

The view BallotCounts produces the count of the number of read and counted Ballots for each ReportingContext.

```
create view BallotCounts (ReportingContext, Read, Counted) as
  select Name, count(BallotId), count (nullif (Accepted, false))
    from Ballot
      natural join ReportingContextAssociationMerge
      right outer join ReportingContext on (Name = ReportingContext)
    group by Name;
```

Assertion 12 *For each ReportingContext, BallotCounts contains exactly one row giving the number of relevant Ballots (in the Read column) and the number of accepted Ballots (in the Counted column).*

1. Using Assertion 3, for each ReportingContext, the natural join of Ballot and ReportingContextAssociationMerge contains one row for each relevant Ballot.

2. For each ReportingContext having zero relevant Ballots, the right outer join with ReportingContext generates a single row having a nulls in the BallotId and Accepted columns. In all other cases, the right outer join has the effect of duplicating the ReportingContext column in the Name column and leaving the other columns unchanged.

3. In the three-way join, there is at least one row for each ReportingContext. Grouping by Name therefore yields one row for each ReportingContext.

4. The count operation does not include nulls. Therefore, for each ReportingContext having one or more relevant Ballots, count(BallotId) yields the number of such Ballots, and count (nullif (Accepted, false)) yields the number of those that were accepted.

5. For each ReportingContext having zero relevant Ballots, the three-way join contains a single row with nulls in both the BallotId and Accepted columns, so both counts yield 0.

4.1.8 BallotCountsByConfiguration

The view BallotCountsByConfiguration produces the count of the number of read and counted Ballots broken down by BallotStyle within each ReportingContext. Rows pertaining to combinations of ReportingContext and BallotStyle that have no applicable Ballots are suppressed.

```
create view BallotCountsByConfiguration (ReportingContext, StyleId,
                                         Read, Counted) as
  select ReportingContext, StyleId, count(*), count (nullif (Accepted, false))
    from Ballot natural join ReportingContextAssociationMerge
    group by ReportingContext, StyleId;
```

Assertion 13 *For each combination of ReportingContext and BallotStyle having relevant Ballots, BallotCountsByConfiguration contains exactly one row giving the number of relevant Ballots (in the Read column) and the number of accepted Ballots (in the Counted column).*

1. Using Assertion 3, for each ReportingContext, the natural join of Ballot and ReportingContextAssociationMerge contains one row for each relevant Ballot.
2. Grouping by ReportingContext and StyleId yields exactly one row for each combination of ReportingContext and BallotStyle having relevant Ballots.
3. count(*) and count (nullif (Accepted, false)) yield the number of Ballots and the number of accepted Ballots within each group.

Assertion 14 *BallotCountsByConfiguration contains no rows for combinations of ReportingContext and BallotStyle that have no relevant Ballots.*

The natural join of Ballot with ReportingContextAssociationMerge cannot generate a row with a particular combination of ReportingContext and BallotStyle unless there exists a Ballot of that BallotStyle that is relevant in that ReportingContext.

4.1.9 BallotCountsByCategory

```
create view BallotCountsByCategory (ReportingContext, Category,
                                    Read, Counted) as
  select ReportingContext, Category, count(*), count (nullif (Accepted, false))
    from Ballot
      natural join ReportingContextAssociationMerge
      natural join BallotCategoryAssociation
    group by ReportingContext, Category;
```

Assertion 15 *For each combination of ReportingContext and BallotCategory having relevant Ballots, BallotCountsByCategory contains exactly one row giving the number of relevant Ballots (in the Read column) and the number of accepted Ballots (in the Counted column).*

1. Using Assertion 3, for each ReportingContext, the natural join of Ballot and ReportingContextAssociationMerge contains one row for each relevant Ballot.
2. For each ReportingContext, for each relevant Ballot, the natural join of the previous result with BallotCategoryAssociation yields a row for each associated BallotCategory.
3. Grouping by ReportingContext and Category yields exactly one row for each combination of ReportingContext and BallotCategory having relevant Ballots.
4. count(*) and count (nullif (Accepted, false)) yield the number of Ballots and the number of accepted Ballots within each group.

Assertion 16 *BallotCountsByCategory contains no rows for combinations of ReportingContext and BallotCategory that have no relevant Ballots.*

The natural join of (Ballot natural join ReportingContextAssociationMerge) with BallotCategoryAssociation cannot generate a row with a particular combination of ReportingContext and BallotCategory unless there exists a Ballot of that BallotCategory that is relevant in that ReportingContext.

4.1.10 BallotCountsByCategoryAndConfiguration

```
create view BallotCountsByCategoryAndConfiguration (ReportingContext, StyleId,
                                                    Category, Read, Counted) as
  select ReportingContext, StyleId, Category, count(*),
         count (nullif (Accepted, false))
    from Ballot
      natural join ReportingContextAssociationMerge
      natural join BallotCategoryAssociation
    group by ReportingContext, StyleId, Category;
```

The assertions and discussion for BallotCountsByCategoryAndConfiguration are analogous to those of BallotCountsByConfiguration and BallotCountsByCategory.

4.1.11 BlankBallot

```
create view BlankBallot (BallotId, StyleId, Accepted) as
  select BallotId, StyleId, Accepted
    from Ballot natural left outer join VoterInput
    where Value is null;
```

Assertion 17 *BlankBallot contains exactly one row for each Ballot having no associated votes.*

Ballot natural left outer join VoterInput produces one row with a non-null Value column for each vote (possibly many such rows with the same BallotId), and exactly one row with a null Value column for each Ballot having no associated votes. The where clause selects only the latter rows.

4.1.12 BlankBallotCounts

```
create view BlankBallotCounts (ReportingContext, Read, Counted) as
  select Name, count(BallotId), count (nullif (Accepted, false))
    from BlankBallot
      natural join ReportingContextAssociationMerge
      right outer join ReportingContext on (Name = ReportingContext)
    group by Name;
```

The assertion and discussion for BlankBallotCounts are parallel to those of BallotCounts, substituting BlankBallot for Ballot.

4.1.13 BlankBallotCountsByConfiguration

```
create view BlankBallotCountsByConfiguration (ReportingContext, StyleId,
                                              Read, Counted) as
  select ReportingContext, StyleId, count(*), count (nullif (Accepted, false))
    from BlankBallot natural join ReportingContextAssociationMerge
    group by ReportingContext, StyleId;
```

The assertions and discussion for BlankBallotCountsByConfiguration are parallel to those of BallotCountsByConfiguration, substituting BlankBallot for Ballot.

4.2 Adaptation

Converting the raw voter inputs into the effective voter inputs required by the logic model involves Alias reconciliation, implementation of straight party voting, and generation of default (0) values for ballot positions that were not voted.

The VoterInput table has a primary key on (BallotId, ChoiceId), so there is at most one row for any given ballot position on any given Ballot. Deliberately, the adaptation views do not preserve this constraint in the event that double votes result from Alias reconciliation or straight party voting. Both of these cases are treated as errors for testing purposes, and the errors are most easily located by looking for duplicate keys. This is done by the integrity view DoubleVotes (see Section 4.3).

All assertions have as a precondition the assumption that all of the constraints are satisfied (see Section 3.4.5).

4.2.1 AntiAliasedVoterInput

AntiAliasedVoterInput provides a view of VoterInput in which all Choices have been "canonicalized."

```
create view AntiAliasedVoterInput (BallotId, ChoiceId, Value) as
  select BallotId, coalesce (Alias.ChoiceId, VoterInput.ChoiceId), Value
    from VoterInput left outer join Alias
      on VoterInput.ChoiceId = Alias.AliasId;
```

Assertion 18 *AntiAliasedVoterInput contains exactly one row for each row in VoterInput, with any non-canonical Choices replaced by the associated canonical Choices.*

1. Alias.AliasId is a primary key, so there can be at most one row in Alias matching any given row of VoterInput on VoterInput.ChoiceId = Alias.AliasId.

2. If there is a row in Alias matching a row of VoterInput, the left outer join generates exactly one row with the canonical equivalent of VoterInput.ChoiceId in the column Alias.ChoiceId.

3. If there is no row in Alias matching a row of VoterInput, the left outer join generates exactly one row with a null in the column Alias.ChoiceId.

4. coalesce (Alias.ChoiceId, VoterInput.ChoiceId) substitutes the canonical Choice in the case where a matching row in Alias exists, and retains the original Choice in the case where no such row exists and Alias.ChoiceId is null.

4.2.2 VoterInputMerge

VoterInputMerge provides a view over AntiAliasedVoterInput in which the side-effects implied by straight party votes have been incorporated. If a straight party selection Contest is overvoted, it has no side-effects.

VoterInputMerge depends on two intervening views, ValidStraightPartyVotes and ImpliedStraightPartyVotes.

```
create view ValidStraightPartyVotes (BallotId, Party) as
  select BallotId, max(Name)      -- There can be only one.  See below.
    from AntiAliasedVoterInput
      natural join Choice
      natural join Contest
    where CountingLogic = 'Straight party selection'
    group by BallotId
    having sum(Value) = 1;        -- There can be only one.
```

Assertion 19 *ValidStraightPartyVotes contains exactly one row for each Ballot containing a vote in a straight party selection Contest, excluding those that overvoted the straight party selection Contest.*

1. The primary key on VoterInput and Assertion 18 ensure that AntiAliasedVoterInput natural join Choice will have the same number of rows as VoterInput.
2. The natural join of the previous result with Contest again adds columns but yields the same number of rows as VoterInput.
3. The where clause eliminates all rows pertaining to Contests that are not straight party selection Contests.
4. Constraint VII states that a given BallotStyle may contain at most one Contest with CountingLogic = Straight party selection.
5. Straight party Contests are implicitly 1-of-M Contests.
6. Constraint I states that any remaining rows must have a VoterInput.Value of 1.
7. The having clause therefore eliminates rows pertaining to straight party Contests that were overvoted (having two or more associated votes).
8. There can be at most one row remaining for a given BallotId. If such a row exists for a given Ballot, max(Name) retrieves the name of the Party that was selected in the straight party selection Contest. (The name cannot be selected directly due to the intervening group by operation.)

Note that ValidStraightPartyVotes does not filter out Ballots that are not accepted.

```
create view ImpliedStraightPartyVotes (BallotId, ChoiceId, Value) as
  select BallotId, ChoiceId, Value
    from VotableChoices
      natural join Endorsement
      natural join ValidStraightPartyVotes;
```

Assertion 20 *ImpliedStraightPartyVotes contains exactly one row for each vote that is implied by a straight party vote.*

1. Per Assertion 5, for each Ballot, VotableChoices contains exactly one row for each canonical Choice for which the Ballot could contain a vote.
2. The primary key on Endorsement ensures that there is at most one row for a given combination of Party and ChoiceId.

3. Constraint XV states that the Choice referenced by an Endorsement must be canonical.

4. For each Ballot, VotableChoices natural join Endorsement contains one row for each vote implied by any possible straight party vote.

5. The natural join of the previous result with ValidStraightPartyVotes eliminates all rows except those pertaining to actual, non-overvoted straight party votes.

Note that ImpliedStraightPartyVotes does not do anything about the possibility that endorsements may be ill-formed (e.g., containing overvotes).

VoterInputMerge simply concatenates the contents of the previous two views. VoterInputMerge does not inherently prevent there from being more than one row with the same (BallotId, ChoiceId). This case violates Constraint XII and is detected by the integrity view StraightPartyOverrides (see Section 4.3).

```
create view VoterInputMerge (BallotId, ChoiceId, Value) as
    select BallotId, ChoiceId, Value from AntiAliasedVoterInput
  union all
    select BallotId, ChoiceId, Value from ImpliedStraightPartyVotes;
```

Assertion 21 *VoterInputMerge contains exactly one row for each row in VoterInput, with any non-canonical Choices replaced by the associated canonical Choices, plus exactly one row for each vote that is implied by a straight party vote.*

The assertion follows from Assertion 18, Assertion 20 and the definition of the union all operator.

Assertion 22 *VoterInputMerge contains at most one row for a given combination of (BallotId, ChoiceId). The Value column contains either (1) the Value derived from a vote for that Choice, (2) the Value derived from a vote for an Alias of that Choice, or (3) the Value derived from a vote for that Choice implied by a straight party vote.*

1. By Assertion 18, Assertion 20, Constraint XII and Constraint XIV, it is not possible for AntiAliasedVoterInput and ImpliedStraightPartyVotes to both contain votes in a given Contest for a given Ballot.

2. Table Choice is defined such that a Choice is uniquely associated with exactly one Contest.

3. Consequently, AntiAliasedVoterInput and ImpliedStraightPartyVotes cannot both contain votes with a given combination of (BallotId, ChoiceId).

4. If a given combination of (BallotId, ChoiceId) appears in AntiAliasedVoterInput, the Value column of VoterInputMerge will contain the Value derived from a vote for the Choice or an Alias of that Choice, satisfying (1) or (2).

5. If a given combination of (BallotId, ChoiceId) appears in ImpliedStraightPartyVotes, the Value column of VoterInputMerge will contain the Value derived from a vote for that Choice implied by a straight party vote, satisfying (3).

4.2.3 EffectiveInput

Finally, the view EffectiveInput generates zeroes for ballot positions that were not voted. This is appropriate for all Contest types except ranked order. Ranked order tabulators should instead access VoterInputMerge directly.

```
create view EffectiveInput (BallotId, ChoiceId, Value) as
  select BallotId, ChoiceId, coalesce (Value, 0)
    from VotableChoices natural left outer join VoterInputMerge;
```

Assertion 23 *For each Ballot, EffectiveInput contains exactly one row for each canonical Choice for which the Ballot could contain a vote. The Value column contains either (1) the Value derived from a vote for that Choice, (2) the Value derived from a vote for an Alias of that Choice, (3) the Value derived from a vote for that Choice implied by a straight party vote, or (4) the default value of zero.*

1. Per Assertion 5, for each Ballot, VotableChoices contains exactly one row for each canonical Choice for which the Ballot could contain a vote.
2. Per Assertion 22, VoterInputMerge contains at most one row for a given combination of (BallotId, ChoiceId).
3. Consequently, VotableChoices natural left outer join VoterInputMerge contains the same number of rows as VotableChoices.
4. Since VotableChoices does not have a Value column, the Value column of the join will come from VoterInputMerge when a matching row exists and be null when no matching row exists.
5. If Value is not null, then it came from VoterInputMerge. Per Assertion 22, this Value will satisfy one of (1), (2), or (3).
6. coalesce (Value, 0) has the effect of replacing nulls with zeroes, satisfying (4).

4.3 Integrity checks

For those integrity constraints that are too complex to code directly as SQL constraints within the tables, a series of views exists to look for problems. All of the integrity checking views should always be empty. If data appear in any of the views, the input was invalid and the results of the model will be invalid. The integrity views are listed in Table 7 but their definitions have been elided. Interested readers can find them in the Votetest distribution, in the file Infrastructure-VoteSchema.sql.

4.4 Translation of logic model

The following transforms are used to render the VVSG 2.0 logic model as SQL.

1. Each function is replaced by a view in which the parameters form the primary key and the last column is the value of the function.
2. Time parameters (t) are factored out. All views implicitly project results for the time t corresponding to the current state of the database.

Table 7: Integrity checks

Constraint	Integrity view(s)
Constraint I	OutOfRangeVoterInputs, OutOfRangeEndorsements
Constraint II	N/A, enforced by SQL check constraint
Constraint III	N/A, enforced by SQL check constraint
Constraint IV	N/A, enforced by SQL check constraint
Constraint V	UnreportedBallots
Constraint VI	ExtraneousInputs
Constraint VII	MoreThanOneStraightPartyContest
Constraint VIII	CircularStraightPartyEndorsements
Constraint IX	NonExistentParties
Constraint X	N/A, enforced by SQL primary key constraint
Constraint XI	DoubleVotes
Constraint XII	StraightPartyOverrides
Constraint XIII	DoubleIndirectAliases
Constraint XIV	CrossContestAliases
Constraint XV	EndorsedAliases
Constraint XVI	TooManyWriteIns

3. When a function takes both a Contest and a Choice as parameters, the Contest parameter is omitted. With the data model used here, the Contest can be inferred from the Choice.

4. Logic is translated into those SQL constructs that are most transparently equivalent.

5. Ranked order Contests, which are not handled by the logic model, are suppressed.

6. Irrelevant values, such as zero tallies for Choices that do not appear in the applicable BallotStyle or Contests that are not relevant in the applicable ReportingContext, are suppressed.

The following subsections first quote relevant portions of the logic model, then describe their analogs in the schema. Some terms from the logic model are elided from this discussion. For complete information on the logic model, please refer to [3].

All assertions have as a precondition the assumption that all of the constraints are satisfied (see Section 3.4.5).

4.4.1 $S(c, r, t, v)$

> Ballot v's vote with respect to contest choice c in contest r as of time t. For checkboxes and the like, the value is 1 (selected) or 0 (not selected). For cumulative voting, the value is the number of votes that v gives to contest choice c in contest r. If the applicable ballot style does not include contest r, $S(c, r, t, v) = 0$.

Assertion 24 *The relevant case of the quaternary function S is implemented by the view EffectiveInput. For each Ballot, for each canonical Choice appearing on that Ballot, column Value contains the value specified by the definition of $S(c, r, t, v)$ for c = ChoiceId, r = Choice.ContestId, t as of the state of the database, and v = BallotId.*

Follows from Assertion 23. Rows are not generated for the irrelevant case where $S(c, r, t, v)$ is defined to be zero. Note that EffectiveInput does not eliminate inputs for ranked order Contests.

4.4.2 $S(r, t, v)$

The total number of votes that ballot v has in contest r as of time t.

$$S(r, t, v) = \sum_{c \in C(r,t)} S(c, r, t, v)$$

The ternary function S is implemented by the view S. The current value of $S(r, t, v)$ is obtained by selecting S_val where ContestId $= r$ and BallotId $= v$. The view S contains rows only for Contests that actually appear on the Ballot according to its BallotStyle. All others are defined to be 0.

```
create view S (ContestId, BallotId, S_val) as
  select ContestId, BallotId, coalesce (sum (Value), 0)
    from FilteredBallotContestAssociation
      natural left outer join Choice
      natural left outer join EffectiveInput
    group by ContestId, BallotId;
```

Assertion 25 *For each Ballot, for each non-ranked-order Contest appearing on that Ballot, column S_val of S contains the value specified by the definition of $S(r, t, v)$ for r = ContestId, t as of the state of the database, and v = BallotId.*

1. Per Assertion 1, for each Ballot, FilteredBallotContestAssociation contains exactly one row for each non-ranked-order Contest appearing in the BallotStyle identified by Ballot.StyleId, and zero rows for any ranked order Contests.

2. Per Assertion 23, for each Ballot, EffectiveInput contains exactly one row for each canonical Choice for which the Ballot could contain a vote.

3. For each Ballot, FilteredBallotContestAssociation natural left outer join Choice contains exactly one row for each Choice in each non-ranked-order Contest on the Ballot, plus exactly one row for each non-ranked-order Contest on the Ballot that has zero associated Choices.

4. For each Ballot, the natural left outer join of the previous result with EffectiveInput contains (1) exactly one row for each canonical Choice in each non-ranked-order Contest on the Ballot, with Value from EffectiveInput, plus (2) exactly one row for each non-canonical Choice in each non-ranked-order Contest on the Ballot, with null Value, plus (3) exactly one row for each non-ranked-order Contest on the Ballot that has zero associated Choices, with null Value.

5. For Contests having Choices, the value specified by the definition of $S(r, t, v)$ follows directly by summing the Value column while grouping by ContestId and BallotId. The null values for non-canonical Choices are ignored by the summation operator. Constraint XIII and Constraint XIV prevent any Contest from having only Aliases as Choices.

6. For Contests with zero associated Choices, the summation of a single null value returns null and coalesce (sum (Value), 0) substitutes the value zero.

VotesByContestAndContext is a convenience to retrieve all of the S_val vote counts for each relevant combination of ReportingContext and Contest. For each relevant combination of ReportingContext and Contest that contains no Ballots, there is a single row with nulls in the last three columns.

```
create view VotesByContestAndContext (ContestId, N, ReportingContext,
                                      BallotId, Accepted, S_val) as
  select ContestId, N, ReportingContext, BallotId, Accepted, S_val
    from FilteredContextContestAssociation
      natural join Contest
      natural left outer join (S natural join ReportingContextAssociationMerge)
      natural left outer join Ballot;
```

The discussion for the following two assertions is combined.

Assertion 26 *For each relevant combination of ReportingContext and Contest (excluding ranked order), VotesByContestAndContext contains a row for each Ballot that contains that Contest and that is reported in that ReportingContext. Column Accepted is as specified in the table Ballot. Column S_val contains the value specified by the definition of $S(r, t, v)$ for r = ContestId, t as of the state of the database, and v = BallotId.*

Assertion 27 *For each relevant combination of ReportingContext and Contest (excluding ranked order) where there does not exist a Ballot that contains that Contest and that is reported in that ReportingContext, VotesByContestAndContext contains a single row with nulls in the BallotId, Accepted, and S_val columns.*

1. Per Assertion 10, for each ReportingContext, FilteredContextContestAssociation contains exactly one row for each non-ranked-order Contest that is relevant in that ReportingContext, and zero rows for each ranked-order Contest.

2. The natural join with Contest adds the column N but does not change the number of rows.

3. Per Assertion 3, for each Ballot, ReportingContextAssociationMerge contains exactly one row for each relevant ReportingContext. Equivalently, for each ReportingContext, ReportingContextAssociationMerge lists every Ballot that is reported in that ReportingContext.

4. Per Assertion 25, for each Ballot, the view S contains exactly one row for each non-ranked-order Contest appearing on that Ballot, with S_val as described above.

5. For each Ballot, S natural join ReportingContextAssociationMerge yields the Cartesian product of the S rows (one for each non-ranked-order Contest appearing on that Ballot) with the ReportingContextAssociationMerge rows (one for each ReportingContext relevant to that Ballot).

6. For each relevant combination of ReportingContext and Contest (excluding ranked order), the natural left outer join of (FilteredContextContestAssociation natural join Contest) and (S natural join ReportingContextAssociationMerge) contains a row for each Ballot that contains that Contest and that is reported in that ReportingContext. For each relevant combination of ReportingContext and Contest (excluding ranked order) where there does not exist a Ballot that contains that Contest and that is reported in that ReportingContext, the result contains a single row with nulls in the BallotId and S_val columns.

7. The natural left outer join with Ballot adds the Accepted column but does not change the number of rows. Where BallotId is null, Accepted is also null.

4.4.3 $S'(c, r, t, v)$

Ballot v's vote with respect to contest choice c in contest r as accepted for counting purposes (i.e., valid votes only), as of time t.

$$t \geq t_E \rightarrow S'(c, r, t, v) = \begin{cases} S(c, r, D(v), v) & \text{if } S(r, D(v), v) \leq N(r) \wedge A(t, v) \\ 0 & \text{otherwise} \end{cases}$$

The quaternary function S' is implemented by the view SPrime. The current value of $S'(c, r, t, v)$ is obtained by selecting SPrime_val where ChoiceId $= c$ and BallotId $= v$.

```
create view SPrime (ChoiceId, BallotId, SPrime_val) as
  select ChoiceId, BallotId,
    case
      when S_val <= N and Accepted then Value
      else 0
    end
    from EffectiveInput
      natural join Choice
      natural join Contest
      natural join Ballot
      natural join S;
```

Assertion 28 *For each Ballot, for each canonical Choice for which that Ballot could contain a vote, excluding ranked order Contests, column SPrime_val of SPrime contains the value specified by the definition of $S'(c, r, t, v)$ for c = ChoiceId, r = Choice.ContestId, t as of the state of the database, and v = BallotId.*

1. Per Assertion 24, the quaternary function S is implemented by the view EffectiveInput. Per Assertion 23, EffectiveInput contains exactly one row for each canonical Choice for which the Ballot could contain a vote.

2. The natural joins with Choice, then Contest, then Ballot add columns but do not change the number of rows.

3. The natural join with S eliminates rows pertaining to Choices in ranked order Contests and adds the S_val column (providing $S(r, t, v)$) to the others.

4. For each Ballot, for each canonical Choice for which the Ballot could contain a vote, the case statement generates the value of $S'(c, r, t, v)$ as specified above.

4.4.4 $T(c, j, r, t)$

The vote total for contest choice c in contest r and reporting context j as of time t. This does not include votes that are invalid due to overvoting or votes from ballots for which $A(t, v)$ is false.

$$t \geq t_E \rightarrow T(c, j, r, t) = \sum_{v \in V(j, t_E)} S'(c, r, t_E, v)$$

The quaternary function T is implemented by the view T. The current value of $T(c, j, r, t)$ is obtained by selecting T_val where ChoiceId = c and ReportingContext = j.

```
create view T (ChoiceId, ReportingContext, T_val) as
  select ChoiceId, ReportingContext, coalesce (sum (SPrime_val), 0)
    from FilteredContextChoiceAssociation
      natural left outer join
        (SPrime natural join ReportingContextAssociationMerge)
    group by ChoiceId, ReportingContext;
```

Assertion 29 *For each ReportingContext, for each canonical Choice in each non-ranked-order Contest that is relevant in that ReportingContext, column T_val of T contains the value specified by the definition of $T(c, j, r, t)$ for c = ChoiceId, j = ReportingContext, r = Choice.ContestId, and t as of the state of the database.*

1. Per Assertion 11, for each ReportingContext, FilteredContextChoiceAssociation contains exactly one row for each canonical Choice in each non-ranked-order Contest that is relevant in that ReportingContext.

2. Per Assertion 28, for each Ballot, for each canonical Choice for which that Ballot could contain a vote, excluding ranked order Contests, column SPrime_val of SPrime contains the value specified by the definition of $S'(c, r, t, v)$.

3. Per Assertion 3, for each Ballot, ReportingContextAssociationMerge contains exactly one row for each relevant ReportingContext.

4. For each Ballot, SPrime natural join ReportingContextAssociationMerge yields the Cartesian product of the SPrime rows with the ReportingContextAssociationMerge rows.

5. For each ReportingContext, the natural left outer join of FilteredContextChoiceAssociation with the previous result adds a single row with a null in the SPrime_val column for each canonical Choice in each non-ranked-order Contest for which there were no relevant Ballots. The rows already existing from the previous result are not changed.

6. For each ReportingContext, for each canonical Choice in each non-ranked-order Contest that is relevant in that ReportingContext and for which at least one relevant Ballot exists, the value specified by the definition of $T(c, j, r, t)$ follows directly by summing the SPrime_val column while grouping by ChoiceId and ReportingContext.

7. For the case in which no relevant Ballot exists, the null that is returned by the summation operation on the single null value is changed to zero by coalesce (sum (SPrime_val), 0).

TSum is a convenience that sums T_val by Contest. Note that TSum does not eliminate ranked order contests but rather provides a value of zero for them in the TSum_val column.

```
create view TSum (ContestId, ReportingContext, TSum_val) as
  select ContestId, ReportingContext, coalesce (sum (T_val), 0)
    from ReportingContextContestAssociation
      natural left outer join Choice
      natural left outer join T
    group by ContestId, ReportingContext;
```

Assertion 30 *For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, column TSum_val of TSum contains the value* $\sum_{c \in C(r,t)} T(c, j, r, t)$ *for* $j =$ *ReportingContext,* $r =$ *ContestId and* t *as of the state of the database.*

1. Per Assertion 9, for each ReportingContext, ReportingContextContestAssociation contains exactly one row for each Contest that is relevant in that ReportingContext.

2. For each ReportingContext, ReportingContextContestAssociation natural left outer join Choice contains one row for each Choice in each Contest that is relevant in that ReportingContext, plus one row with null in the ChoiceId column for each Contest that is relevant in that ReportingContext that has no associated Choices.

3. The natural left outer join of the previous result with T, using both the ChoiceId and ReportingContext columns, has the same number of rows as the previous result. Each row pertaining to a Choice in some non-ranked-order Contest acquires T_val supplying the value of $T(c, j, r, t)$. Each row pertaining to a ranked order Contest or a Contest with no associated Choices acquires T_val containing a null value.

4. For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext and that has at least one associated Choice, the value specified by $\sum_{c \in C(r,t)} T(c, j, r, t)$ follows directly by summing the T_val column while grouping by ContestId and ReportingContext.

5. For ranked order Contests and Contests having no associated Choices, the null that is returned by the summation operation on the single null value is changed to zero by coalesce (sum (T_val), 0).

4.4.5 $O(j, r, t)$

For a given contest and reporting context, the number of overvotes in read ballots for which $A(t, v)$ is true as of time t. Each ballot in which contest r is overvoted contributes $N(r)$ to $O(j, r, t)$.

$$t \geq t_E \rightarrow O(j, r, t) = \sum_{v \in V(j, t_E)} \begin{cases} N(r) & \text{if } S(r, D(v), v) > N(r) \wedge A(t, v) \\ 0 & \text{otherwise} \end{cases}$$

The ternary function O is implemented by the view O. The current value of $O(j, r, t)$ is obtained by selecting O_val where ContestId $= r$ and ReportingContext $= j$.

```
create view O (ContestId, ReportingContext, O_val) as
  select ContestId, ReportingContext, sum (
      case
        when S_val > N and Accepted then N
        else 0
      end                                    )
    from VotesByContestAndContext
    group by ContestId, ReportingContext;
```

Assertion 31 *For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, column O_val of O contains the value specified by the definition of* $O(j, r, t)$ *for* $j =$ *ReportingContext,* $r =$ *ContestId, and* t *as of the state of the database.*

1. Per Assertion 26, for each relevant combination of ReportingContext and Contest (excluding ranked order), VotesByContestAndContext contains a row for each Ballot that contains that Contest and that is reported in that ReportingContext. Column Accepted is as specified in the table Ballot. Column S_val contains the value specified by the definition of $S(r, t, v)$ for $r =$ ContestId, t as of the state of the database, and $v =$ BallotId.

2. Per Assertion 27, for each relevant combination of ReportingContext and Contest (excluding ranked order) where there does not exist a Ballot that contains that Contest and that is reported in that ReportingContext, VotesByContestAndContext contains a single row with nulls in the BallotId, Accepted, and S_val columns.

3. For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, the value specified by the definition of $O(j, r, t)$ follows directly by summing the result of the case statement for each row while grouping by ContestId and ReportingContext. In the case where Accepted and S_val are null, i.e., where there does not exist a Ballot that contains that Contest and that is reported in that ReportingContext, the case statement returns 0.

4.4.6 $U(j, r, t)$

For a given contest and reporting context, the number of undervotes in read ballots for which $A(t, v)$ is true as of time t. A given ballot contributes at most $N(r)$ to $U(j, r, t)$. Ballot styles that do not include contest r do not contribute to this total.

$$t \geq t_E \rightarrow U(j, r, t) = \sum_{v \in V(j, t_E)} \begin{cases} N(r) - S(r, D(v), v) & \text{if } S(r, D(v), v) \leq N(r) \wedge A(t, v) \\ 0 & \text{otherwise} \end{cases}$$

The ternary function U is implemented by the view U. The current value of $U(j, r, t)$ is obtained by selecting U_val where ContestId $= r$ and ReportingContext $= j$.

```
create view U (ContestId, ReportingContext, U_val) as
  select ContestId, ReportingContext, sum (
      case
        when S_val <= N and Accepted then N - S_val
        else 0
      end                                    )
    from VotesByContestAndContext
    group by ContestId, ReportingContext;
```

Assertion 32 *For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, column U_val of U contains the value specified by the definition of $U(j, r, t)$ for $j =$ ReportingContext, $r =$ ContestId, and t as of the state of the database.*

The argument is parallel to that of Assertion 31, substituting U for O.

4.4.7 $K(j, r, t)$

For a given contest and reporting context, the number of read ballots for which $A(t, v)$ is true as of time t (i.e., the number of ballots that should be counted). Ballot styles that do not include contest r do not contribute to this total.

The ternary function K is implemented by the view K. The current value of $K(j, r, t)$ is obtained by selecting K_val where ContestId $= r$ and ReportingContext $= j$.

```
create view K (ContestId, ReportingContext, K_val) as
  select ContestId, ReportingContext,
    (select count(*)
       from Ballot
         natural join ReportingContextAssociationMerge
         natural join BallotStyleContestAssociation
       where ReportingContextAssociationMerge.ReportingContext
         = FilteredContextContestAssociation.ReportingContext
       and BallotStyleContestAssociation.ContestId
         = FilteredContextContestAssociation.ContestId
       and Accepted)
    from FilteredContextContestAssociation;
```

Assertion 33 *For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, column K_val of K contains the value specified by the definition of $K(j, r, t)$ for $j =$ ReportingContext, $r =$ ContestId, and t as of the state of the database.*

1. Per Assertion 10, for each ReportingContext, FilteredContextContestAssociation contains exactly one row for each non-ranked-order Contest that is relevant in that ReportingContext, and zero rows for each ranked-order Contest.

2. Per Assertion 3, for each Ballot, ReportingContextAssociationMerge contains exactly one row for each relevant ReportingContext.

3. For each Ballot, Ballot natural join ReportingContextAssociationMerge contains exactly one row for each relevant ReportingContext.

4. For each Ballot, for each relevant ReportingContext, the natural join of the previous result with BallotStyleContestAssociation contains exactly one row for each Contest appearing on the Ballot. If the BallotStyle has no associated Contests, there are zero such rows.

5. For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, the value specified by the definition of $K(j, r, t)$ follows directly from counting the number of rows in the previous three-way join that have a matching ReportingContext and ContestId, and where Accepted is true.

4.4.8 Balance

Every vote must be accounted for.

$$t \geq t_E \rightarrow \sum_{c \in C(r,t)} T(c, j, r, t) + O(j, r, t) + U(j, r, t) = K(j, r, t) \times N(r)$$

A check for this assertion is implemented by the view Balance. The current difference between $\sum_{c \in C(r,t)} T(c,j,r,t) + O(j,r,t) + U(j,r,t)$ and $K(j,r,t) \times N(r)$ is obtained by selecting Discrepancy where ContestId $= r$ and ReportingContext $= j$. Discrepancy should always be zero.

```
create view Balance (ContestId, ReportingContext, Discrepancy) as
  select ContestId, ReportingContext, K_val * N - (TSum_val + O_val + U_val)
    from K
      natural join TSum
      natural join O
      natural join U
      natural join Contest;
```

Assertion 34 *For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, column Discrepancy of Balance contains the value* $K(j,r,t) \times N(r) - \left(\sum_{c \in C(r,t)} T(c,j,r,t) + O(j,r,t) + U(j,r,t)\right)$ *for* $j =$ *ReportingContext,* $r =$ *ContestId, and* t *as of the state of the database.*

1. Per Assertion 30, Assertion 31, Assertion 32 and Assertion 33, the views K, TSum, O and U each provide one of the needed values for each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext. (TSum additionally provides rows for ranked order Contests that will be eliminated.)

2. The successive natural joins of K, TSum, O and U all occur on the columns ContestId and ReportingContext. Each join adds a column but does not change the number of rows. The rows in TSum pertaining to ranked order contests are eliminated by the first join.

3. The natural join with Contest (on the column ContestId) adds the column N but does not change the number of rows.

4. For each ReportingContext, for each non-ranked-order Contest that is relevant in that ReportingContext, the value specified above follows directly from the expression K_val * N − (TSum_val + O_val + U_val).

5 Advanced test development environment

5.1 Software prerequisites

All software, tools and materials were developed on a GNU/Linux operating system.

The following packages are required in order to build and run the programs in Votetest:

Name	Short name	Version tested	Source
PostgreSQL	postgres	8.3.7[4]	[7]
Class Library for Numbers	CLN	1.2.2	[8]
GNU Compiler Collection	g++	4.3.3	[9]
Flex: Fast Lexical Analyzer	flex	2.5.35	[10]
Bison: GNU parser generator	bison	2.3	[11]

[4]Some planner problems with outer joins that interfered with the operation of the schema were fixed in version 8.2.5 (see Bug #3426).

Votetest uses extensions to the SQL standard [2] and the C++ standard [12] that might not function as intended with other databases and compilers.

Some help in configuring PostgreSQL is provided in Section 5.9.

5.2 Hardware prerequisites

Votetest was developed on a PC having a 3.6 GHz Pentium 4 processor, 1 GiB of RAM, and an 80 GB SATA hard drive. The resources of this PC were more than adequate for all tests in the basic test suite, and if necessary a lesser configuration should be usable. Performance limitations became obvious only with the large and complex scenarios generated for the scalability testing described in Section 5.8.

5.3 File listing

Files pertaining to the advanced test development environment are described in Table 8.

Table 8: Advanced files in the Votetest distribution

Files	Description	Details
runTest	Shell script to execute a test case.	Section 5.5.1
runAllTests	Shell script to execute all test cases and save output.	Section 5.5.2
output/	Empty subdirectory used to store the output of runAllTests.	Section 5.5.2
output_kill-overvotes/	Empty subdirectory used to store the output of runAllTests when the –kill-overvotes option is used.	Section 5.5.2
0-integrity-*description*.sql	SQL, test suite integrity checks (15 files).	Section 5.6
Infrastructure-Features.sql	SQL, create table mapping test cases to system capabilities.	Section 5.5.4
Infrastructure-IntegrityChecks.sql	SQL, show contents of all integrity views. This is invoked automatically by test cases and need not be used directly.	N/A
Infrastructure-KillOvervotes.sql	SQL, transform a test case into one that is executable on a system that prevents overvoting. This is invoked automatically by test cases and need not be used directly.	N/A
Infrastructure-PairsCoverage.pgcc	Source code of coverage checking utility.	Section 5.5.5
Infrastructure-PairsCoverage.sql	SQL, data needed by coverage checking utility.	Section 5.5.5
Infrastructure-TestFooter.sql	SQL, print “END TEST CASE OUTPUT” footer. This is invoked automatically by test cases and need not be used directly.	N/A

Infrastructure-TestHeader.sql	SQL, print "BEGIN TEST CASE OUTPUT" and timestamp and configure verbosity of output for all test cases. This is invoked automatically by test cases and need not be used directly.	N/A
Infrastructure-TestHook.sql	SQL, support the optional invocation of a test transformation script (e.g., Infrastructure-KillOvervotes.sql). This is invoked automatically by test cases and need not be used directly.	N/A
Infrastructure-VoteSchema.sql	SQL, create the schema. This is invoked automatically by test cases and need not be used directly.	Section 3.5 Section 4
ReportGenerator/	Subdirectory containing the source code of the advanced test development environment's report generator. This is invoked automatically by test cases and need not be used directly.	Section 5.5.3
TestGenerator/	Subdirectory containing the source code of the advanced test development environment's test generator.	Section 5.7
test_specs/	Subdirectory containing example input files for the test generator.	Section 5.7.3
Makefile.am Makefile.in aclocal.m4 configure configure.ac depcomp install-sh missing	Files having to do with the automake build process.	Section 5.4
INSTALL	Text file containing generic instructions on the use of the configure script.	Section 5.4
AUTHORS NEWS README	Unused files required by GNU standard.	N/A

5.4 Installation

Votetest is packaged with GNU automake [13], so all usual GNU tricks should work. Help on configuration options can be found in the INSTALL file or obtained by entering `./configure --help`.

Normally, one should only need to do the following to compile ReportGenerator and TestGenerator.

```
bash-3.1$ ./configure
bash-3.1$ make
```

However, in the event that PostgreSQL and/or CLN are installed in nonstandard locations, an invocation such as the following might be required.

```
bash-3.1$ ./configure \
> CPPFLAGS="-I/usr/local/pgsql/include -I/usr/local/cln-1.2.2/include" \
> LDFLAGS="-L/usr/local/pgsql/lib -L/usr/local/cln-1.2.2/lib"
bash-3.1$ make
```

5.5 Infrastructure

5.5.1 runTest

The script runTest is used to run an SQL test case against the database and report the results from the Votetest environment.

A test case is executed by changing the current working directory to the directory containing the test suite and invoking the runTest script with the file name of the test case. The runTest script resets the database to an initial state and then feeds the test case to the SQL interpreter. No database named votetest other than the one created by the test suite should exist or it will be destroyed without warning.

`Usage: ./runTest [--kill-overvotes] test-file-name.sql`

Some test cases involve overvoting, which means they cannot be executed as-is on a system that prevents overvoting. If the –kill-overvotes switch is used, the test case is transformed into one that can be executed on a system that prevents overvoting. Otherwise, overvotes are processed and reported as they would be in a system that supports overvoting.

If a ballot overvotes a contest, –kill-overvotes removes all of that ballot's votes in that contest, effectively converting overvotes into undervotes. This of course changes the expected results of the test case. The expected results in the normal configuration and with –kill-overvotes enabled are saved in the subdirectories sample_output and sample_output_kill-overvotes respectively.

5.5.2 runAllTests

The script runAllTests invokes runTest for each test case and directs the output into a file in the subdirectory named "output." It then invokes runTest –kill-overvotes for each test case and directs the output into a file in the subdirectory named "output_kill-overvotes." These results may then be compared with the contents of the sample_output and sample_output_kill-overvotes subdirectories to ensure that the test suite is operating as expected.

5.5.3 ReportGenerator

Usage: `ReportGenerator [-v] context-name [context-name...]`. A no-frills, plain-ASCII post-voting report for each specified ReportingContext is sent to standard output. As a convenience to test labs, the report total volume needed for the accuracy test protocol of the VVSG is also calculated and reported. A sample report is shown in Figure 9.

Figure 9: Sample report

Report for context Precinct 1 generated 2007-03-21 09:19-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		13	13
	Blank	1	1
Precinct 1 Style		13	13
	Blank	1	1

VOTE TOTALS

Straight party, vote for at most 1	
Bipartisan Party	1
Moderate Party	1
Overvotes	1
Undervotes	10
Counted ballots	13
Balance	0
President, vote for at most 1	
Car Tay Fower	4
Tayra Tree	3
Beeso Tu (Moderate Party)	2
Oona Won (Bipartisan Party)	1
Nada Zayro	0
Overvotes	1
Undervotes	2
Counted ballots	13
Balance	0

Report total volume: 108
- Includes optional reporting of blank ballots.
- Excludes separate reporting of ballots cast vs. read.

Table 9: ReportGenerator return codes

Bit	Meaning
00001	Incorrect usage
00010	No such reporting context
00100	Exception on attempt to connect to database
01000	Exception while connected
10000	Exception on attempt to disconnect from database

The verbose flag (-v) is only useful in ranked order contests. It causes the state of the ranked order logic (with many ballot images) to be output for each round of voting.

ReportGenerator is normally invoked by individual test cases and need not be used directly. If it is invoked from a shell script, the codes that it returns to the shell are listed in Table 9. A return of 0 indicates success; other values indicate one or more problems as encoded by individual bits. Consult the standard error output of the program for additional details on the failure or failures that occurred.

If an error similar to the following occurs when ReportGenerator is invoked:

```
ReportGenerator/ReportGenerator: error while loading shared libraries:
  libecpg.so.5: cannot open shared object file: No such file or directory
```

The solution is to add a command like the following to ~/.bash_profile or another script that is always executed, specifying the location of the library that was not found.

```
export LD_LIBRARY_PATH=/usr/local/pgsql/lib
```

The algorithm used for ranked order contests is only one example of conforming behavior. This algorithm is not recommended or endorsed by the National Institute of Standards and Technology for use in elections and it is probably not the best algorithm available for the purpose. It is used in Votetest only to provide output for comparison in simple cases where the implementation-dependent details have no impact.

- The quota is Hagenbach-Bischoff plus epsilon.
- Surpluses are transferred via the Gregory method using unlimited precision rational numbers.
- No special cases are handled. This means: every Choice must be ranked on every Ballot; every Choice must be assigned a different rank; only one Choice is elected or eliminated at a time; and if a tie occurs, the algorithm halts.

5.5.4 Features

Infrastructure-Features.sql creates the following table, which maps test cases to system capabilities (VVSG classes):

```
create table Features (
  Test                           Text     primary key,

  PrimaryElections               Boolean  not null default false,
  AbsenteeVoting                 Boolean  not null default false,
  AbsenteeByCategories           Boolean  not null default false,
  SplitPrecincts                 Boolean  not null default false,
  BallotRotation                 Boolean  not null default false,
  WriteIns                       Boolean  not null default false,
  CumulativeVoting               Boolean  not null default false,
  NofMVoting                     Boolean  not null default false,
  RankedOrderVoting              Boolean  not null default false,
  ProvisionalChallengedBallots   Boolean  not null default false,
  StraightPartyVoting            Boolean  not null default false,
  CrossPartyEndorsement          Boolean  not null default false,

  check (StraightPartyVoting or not CrossPartyEndorsement),
  check (AbsenteeVoting or not AbsenteeByCategories)
);
```

One could find the set of test cases applicable to a system that lacks support for certain features (e.g., ranked order voting and straight party voting) as follows:

```
bash-3.1$ psql votetest
[... PostgreSQL interactive terminal starts ...]

votetest=# \i Infrastructure-Features.sql
[... Features table is created ...]

votetest=# select Test from Features where not RankedOrderVoting
votetest-#   and not StraightPartyVoting;
[... Test cases are listed ...]
```

5.5.5 PairsCoverage

While the purpose of Features is to recall all test cases that apply for a given set of supported features, the purpose of PairsCoverage is to establish that test cases exist for every pair of features. In PairsCoverage, subclassed voting variations are treated separately to establish that a test case exists for both cases. Since all meaningful pairings are now covered, the PairsCoverage tool is no longer being maintained or updated as new test cases are added.

PairsCoverage is prepared and run as follows:

```
bash-3.1$ psql votetest < Infrastructure-PairsCoverage.sql
bash-3.1$ ./Infrastructure-PairsCoverage > matrix.html
```

Infrastructure-PairsCoverage outputs the content of Table 10 in HTML form. A key to the row and column headings is provided in Table 11.

Table 10: Output of Infrastructure-PairsCoverage

	PE	**AV**	**ABC**	**SP**	**BR**	**WI**	**CV**	**NMV**	**ROV**	**PCB**	**SPV**	**CPE**
PE	12	1	1	1	1	1	1	1	1	1	1	1
AV	1	13	N/A	1	1	2	1	1	1	1	1	1
ABC	1	N/A	12	1	1	2	1	1	1	1	1	1
SP	1	1	1	13	1	1	1	1	1	1	1	1
BR	1	1	1	1	12	1	1	1	1	1	1	1
WI	1	2	2	1	1	16	1	2	1	1	1	1
CV	1	1	1	1	1	1	12	1	1	1	1	1
NMV	1	1	1	1	1	2	1	13	1	1	1	1
ROV	1	1	1	1	1	1	1	1	12	1	1	N/A
PCB	1	1	1	1	1	1	1	1	1	12	1	1
SPV	1	1	1	1	1	1	1	1	1	1	11	N/A
CPE	1	1	1	1	1	1	1	1	N/A	1	N/A	10

Table 11: Key to headings appearing in Table 10

Heading	**Test suite ID**
PE	PrimaryElections
AV	AbsenteeVoting
ABC	AbsenteeByCategories
SP	SplitPrecincts
BR	BallotRotation
WI	WriteIns
CV	CumulativeVoting
NMV	NofMVoting
ROV	RankedOrderVoting
PCB	ProvisionalChallengedBallots
SPV	StraightPartyVoting
CPE	CrossPartyEndorsement

Table 12: Constraint violation tests

Constraint	Test case(s)
Constraint I	0-integrity-OutOfRangeInput.sql, 0-integrity-OutOfRangeEndorsement.sql
Constraint II	N/A, enforced by SQL check constraint
Constraint III	N/A, enforced by SQL check constraint
Constraint IV	N/A, enforced by SQL check constraint
Constraint V	0-integrity-UnreportedBallots.sql
Constraint VI	0-integrity-ExtraneousInput.sql
Constraint VII	0-integrity-MoreThanOneStraightPartyContest.sql
Constraint VIII	0-integrity-CircularEndorsement.sql
Constraint IX	0-integrity-NonExistentParties.sql
Constraint X	N/A, enforced by SQL primary key constraint
Constraint XI	0-integrity-AliasDoubleVotes.sql
Constraint XII	0-integrity-StraightPartyOverrides.sql
Constraint XIII	0-integrity-DoubleIndirectAlias.sql
Constraint XIV	0-integrity-CrossContestAliases.sql
Constraint XV	0-integrity-EndorsedAlias.sql
Constraint XVI	0-integrity-TooManyWriteIns.sql

5.6 Test suite self-tests

Self-tests are for the purpose of validating the test suite itself, not for testing voting systems. The integrity and self-testing features of a voting system may or may not bear any resemblance to the integrity and self-testing features of the test suite.

5.6.1 Baseline

The test case 0-integrity-Baseline.sql verifies that the integrity views show no false positives on the base state for integrity tests.

5.6.2 Constraint violations

The operation of schema constructs designed to detect violations of the constraints specified in Section 3.4.5 is verified by test cases that deliberately violate them. The test cases are listed in Table 12.

5.7 TestGenerator

The test generator is an extra testing tool that may be useful in the creation of additional tests beyond those of the basic test suite, for example, for volume testing. However, it is not part of the basic test suite. A different approach to generating test data is detailed in Section 6.5.6.

```
Usage:  TestGenerator input-filename > output.sql
```

The input to the test generator is a plain text file having the following format:

Input:	ElectionSpec ContestSpec*
ElectionSpec:	NameValuePair+
ContestSpec:	(NameValuePair+)
NameValuePair:	Name = Value

Whitespace, /* C */ and // C++ style comments are ignored.

The permissible names and values are detailed below.

5.7.1 Election specification

The following values must be specified using name-value pairs:

- ballots = number of ballots
- districts = number of districts
- precincts = number of precincts

The following fields have default values and may be omitted:

- ballotDistribution = UniformRandom (default) or Even.
- precinctDistribution = UniformRandom (default) or Even.

The ballot distribution controls the assignment of ballots to precincts. The precinct distribution controls the assignment of precincts to districts.

> **UniformRandom**. X are assigned randomly to Y such that on average all Y would get the name number of X; however, no specific number is aimed for and it is unlikely that all Y will in fact receive exactly the same number of X. The X pertaining to each Y are not grouped (i.e., they do not appear with consecutive numbers in the generated test).
>
> **Even**. X are deterministically divided as evenly as possible across Y and are grouped by Y.

5.7.2 Contest specification

The following values must be specified using name-value pairs:

- N = per definition of N in Section 3.4.3.4 (> 0).
- M = number of choices to generate ($\geq N$).

The following fields have default values and may be omitted:

- level = S (system extent, default), D (district), or P (precinct).
- logic = N (N-of-M, default), C (cumulative), or R (ranked order).
- distribution = UniformRandom (default) or Triangle.

- W = number of write-in choices to generate ($0 \leq W \leq M$, default 0).

If a Contest is declared as system extent level, one Contest that appears on every Ballot is generated. If a Contest is declared as district level, a separate Contest is generated for each district and appears only on Ballots for that district. If a Contest is declared as precinct level, a separate Contest is generated for each precinct and appears only on Ballots for that precinct. The assignment of precincts to districts is random.

Distribution controls how votes from individual ballots are distributed to choices.

> **UniformRandom**. For N-of-M and Cumulative contests, votes are distributed randomly to contest choices such that on average they would get the same number of votes; however, no specific tally is aimed for and it is unlikely that all choices will in fact receive exactly the same number of votes. For ranked order contests, every ballot ranks all of the ballot choices in random order.
>
> **Triangle**. For N-of-M and Cumulative contests, votes are deterministically assigned to contest choices such that, for some integer $X \geq 0$, the first contest choice will receive X votes, the second contest choice will receive $2X$ votes, and so on. X is made as large as possible for the available number of votes and ballots (in an N-of-M contest, the tally for any contest choice cannot exceed the number of ballots). Any surplus votes are left as undervotes. For ranked order contests, every ballot ranks all of the ballot choices in reverse order so that the highest numbered contest choice is elected in the first round of voting, the second highest numbered contest choice is elected in the second round of voting, etc.

If W is nonzero, the first W choices by number become write-ins. Thus, in a Triangle distribution, write-ins get fewer votes than other choices.

5.7.3 Example

```
ballots=200 districts=2 precincts=4
(level=S logic=N N=1 M= 7 distribution=Triangle)
(level=S logic=N N=1 M= 6 distribution=Triangle)
(level=D logic=N N=1 M= 4 distribution=Triangle)
(level=D logic=N N=1 M= 4 distribution=Triangle)
(level=D logic=N N=4 M=10 distribution=Triangle)
(level=P logic=N N=4 M=16 distribution=Triangle)
```

In this example, the distribution of votes is deterministic, but the assignment of ballots to precincts and precincts to districts is random. This randomness affects the number of votes and ballots available in each contest, so the results in each contest will vary as the Triangle distributions scale up or down accordingly.

Additional examples can be found in the test_specs subdirectory of the Votetest distribution.

5.8 On performance and scalability

Votetest was designed with a preference for transparency of logic over performance and scalability. With the exception of ranked order, tabulation logic is implemented in SQL and mirrors the logic model defined in the VVSG. This limits opportunities to introduce faults but incurs a considerable performance penalty.

To improve performance in large and complex test cases, ReportGenerator builds temporary tables corresponding to views that are in or near the top level of the schema. This avoids repeated computation of views that are accessed many times during report generation. Nevertheless, for sufficiently large and complex test cases, the construction of these temporary tables becomes I/O bound and the time to generate reports therefore becomes quite long.

Table 13 shows the run time for several test cases of significant size and complexity as observed on the computer described in Section 5.2. The parameters used in these examples are believed to exceed the limits likely to be used in conformity assessment. Performance will vary by PostgreSQL version and configuration, by hardware configuration, and by workload.

Table 13: Scalability figures

Ballots	Districts	Precincts	Contests	Ballot positions	Votes	Run time
2000000	1	1	1	10000000	6000000	27 min
2000000	5	25	1	10000000	6000000	32 min
2000000	1	1	3	30000000	6000000	1 hr 6 min
120000	4	60	$15 + 4 \times 4 + 60 = 91$	9600000	2400000	26 min

Within reasonable limits, slow execution should not be a barrier to testing. Test cases are typically executed by Votetest only once to obtain results for comparison with those from actual voting system products. The output from every test case provided as part of the distribution is saved in the sample_output and sample_output_kill-overvotes subdirectories. Any new tests that are developed by test labs can similarly be executed in advance and the results saved for future reference.

N.B., The –kill-overvotes test transformation option was not used during the scalability tests. Additional optimization may be required if it is necessary to transform large test cases.

5.9 PostgreSQL configuration help

The following is intended to help someone unfamiliar with PostgreSQL reach a usable configuration with minimal effort. More complete information is available in the PostgreSQL documentation.

The steps to reaching a usable configuration are:

1. Create the Unix account ‘postgres.’
2. Install the PostgreSQL software.
3. Initialize the database.
4. Customize the database configuration.
5. Start the database dæmon.
6. Grant database access to the testing account.

5.9.1 Create the Unix account 'postgres'

This step is performed according to the procedures of the Unix operating system being used. Commonly there is a shell script in /usr/sbin for this purpose, that must be run as root.

The postgres account should be locked to prevent interactive log-ins:

```
bash-3.1# passwd -l postgres
Password changed.
```

5.9.2 Install the PostgreSQL software

If a pre-built PostgreSQL package is not available, PostgreSQL may be built from source code.

```
bash-3.1$ ./configure --prefix=/usr/local
bash-3.1$ make
bash-3.1$ su
bash-3.1# make install
```

5.9.3 Initialize the database

Assuming that the database should go in /usr/share/data:

```
bash-3.1# mkdir /usr/share/data
bash-3.1# chown postgres. /usr/share/data
bash-3.1# su postgres
bash-3.1$ initdb -D /usr/share/data
```

5.9.4 Customize the database configuration

The database configuration is customized by modifying the file 'postgresql.conf' in the data directory (/usr/share/data in this example).

The following changes to the default configuration are recommended for a database being used exclusively for Votetest. They would not necessarily be appropriate for a database that is also used for other purposes.

- fsync = off
- full_page_writes = off
- checkpoint_segments = 50

PostgreSQL supports several different approaches to access control. If the database is to be used in a multi-user or networked environment, please consult the PostgreSQL documentation to determine which access control approach is optimal for your configuration.

5.9.5 Start the database dæmon

```
bash-3.1# touch /var/log/postgres
bash-3.1# chown postgres. /var/log/postgres
bash-3.1# su postgres -c "nohup /usr/local/bin/postgres -D /usr/share/data >&
/var/log/postgres < /dev/null &"
```

The final command should be added to an /etc/rc.d script to automate restarting the database after each system reboot.

5.9.6 Grant database access to the testing account

As user postgres, use the PostgreSQL command-line script 'createuser' to grant database access to the Unix account that will be running Votetest (in this example, johndoe).

```
bash-3.1$ createuser johndoe
Shall the new role be a superuser? (y/n) y
CREATE ROLE
```

5.10 Votetest under Cygwin

It is possible to compile and run the programs of the Votetest distribution under Microsoft Windows using Cygwin [14]. However, this configuration has known problems and has not been thoroughly tested.

Procedures for installing and troubleshooting PostgreSQL under Cygwin, along with some known problems, are documented in the Installing PostgreSQL on Windows Using Cygwin FAQ [15].

Note also the following:

- The version of PostgreSQL that is available as a binary package for Cygwin is usually too old, so the latest version must be built from source.
- If the results of the configure script are used without modification, the PostgreSQL server reports the error `FATAL: setsid() failed: Operation not permitted` at nondeterministic intervals, and the test cases that are in progress at the time fail. To avoid this problem, the file pg_config.h that is produced by the configure script must be patched as follows before PostgreSQL is compiled.

  ```
  sed --in-place --expression="s/#define HAVE_SETSID 1/#undef HAVE_SETSID/" \
      src/include/pg_config.h
  ```

- The PostgreSQL lib directory must be added to PATH. Adding it to LD_LIBRARY_PATH does not work.
- In addition to running cygserver as described in the Installing PostgreSQL on Windows Using Cygwin FAQ, it is necessary to set the environment variable CYGWIN to the value server (`export CYGWIN=server`).

- Some versions of the PostgreSQL server encountered many errors of the form `could not remove file or directory "base/55958": Directory not empty` and leaked directories under (data_directory)/base at a significant rate. As a workaround, it is advisable to re-initialize the entire database at regular intervals to recover the leaked resources.
- According to the Installing PostgreSQL on Windows Using Cygwin FAQ, Cygwin emulates local Unix sockets using Internet sockets that are visible on the network, creating a security hazard.

6 New test case walk-through

6.1 Introduction

To help in understanding the Votetest model, this section presents an example election scenario, maps that example to both the Votetest data model and the database schema that realizes it, and describes how one could proceed to use that example as a testing scenario.

Section 3.4 defines an abstract UML model to represent a potential election and the vote counting capabilities of a voting system. Portions of the abstract model will be used at appropriate times to help in understanding its relationship to the example presented herein. The concrete realization of that model as a relational database schema provided in Section 3.5 will also be used later in this section.

This section begins with a use case that may be analogous to the basic requirements of a local election district and shows how Votetest might be used to help create specific test cases relevant to that election district.

Section 6.2 presents an example election in layman's terms. Section 6.3 explains how the features of that example get represented in the Votetest abstract model. Section 6.4 shows how the details of the example might be represented in the Votetest physical model as tables in an SQL relational database. Section 6.5 gives some assistance in how to use the Votetest model to generate specific test cases, Section 6.6 presents several of the reports generated by Votetest for the sample elections, and Section 6.7 draws some general conclusions.

6.2 Example election

Consider a mid-sized county that serves as the election district for a general election that includes federal, state and local contests. Suppose the county is small enough that the federal contests are the same throughout the county but large enough that the state and local contests vary. Some state contests may be larger than just the county, some may be included entirely within the county, and some may be split across county lines. The local contests are all within the county, but there may be political units within the county that cross voting precincts and have different ballot styles.

The county needs to prepare for several different kinds of contests, including political offices where only a single candidate is the winner, political offices where there are multiple candidates and multiple winners (e.g., County Council seats), and ballot initiatives that are voted either up or down. A small municipality within the county allows cumulative voting for its City Council members, so the voting system for the county will also have to support cumulative voting for that contest.

The Votetest model also supports straight-party voting and ranked order voting, but this county currently does not allow either of those voting styles in any contests. The county will have to support primary elections in the future with several different ballot styles within each precinct, but for simplicity this example is a general election with exactly one ballot style in each precinct.

Individual candidates may be affiliated with a political party and local law requires that for specific offices the party of the candidate be listed alongside their name on the ballot. It's also possible that local political parties may endorse candidates for offices that are considered non-political. In the Votetest model, party affiliations and party endorsements are modelled independently, the first to support labeling of party affiliation on the ballot and the second to support the straight-party voting style popular in some election districts. The independence of these two features allows conflicting affiliations versus endorsements, but that is a feature rather than a bug in the model since cross-party endorsements occur in some elections. This example election does not have any straight-party voting contests, and endorsements will not be printed on the ballot, but the county desires to track such endorsements for possible reporting purposes.

The county has a number of voting precincts and is required to report results by precinct as well as by state delegate districts that fall within the county. In some cases it may desire to report the results of ballot initiatives by geographic or political sub-divisions of the county. Each precinct has a number of voting machines, but all results are totalled within the precinct by a single precinct tabulator. For simplicity, we assume that precincts are the smallest unit requiring result reporting; however, the data model (Section 3.4) is capable of handling precincts split across political units.

By a fluke of election law, write-ins are not supported for federal contests or for some state-wide contests, but they are usually allowed for local offices, including multiple write-ins for offices that have multiple winners. (This fluke is contrived to increase the variety of contest types in the example.) Additionally, the interpretation of write-ins and the crediting of valid write-in votes to persons eligible to hold that office is deferred until after the close of polls, in accordance with jurisdiction policies and procedures.

6.2.1 Federal and statewide contests

- U.S. President—Three candidates, each registered by a recognized political party, no write-ins allowed.
- U.S. Senate—Two candidates, each registered by one of the parties appearing in the presidential contest, no write-ins allowed.
- U.S. House—Three candidates, two registered by the same parties as for U.S. Senate, but the third registered with a 4^{th} party different from any of those above, no write-ins allowed.
- State Senate—Three candidates, registered with the same three parties as for U.S. House, no write-ins allowed.
- State House—Multiple house districts within the county identified below.

6.2.2 Local single winner contests

- State House District #1—Three candidates, registered with the same parties as for U.S. House, ballot must accommodate single potential write-in. District completely within the county.

- State House District #2—Three candidates, two registered with the same parties as U.S. Senate, but the third registered as an Independent, ballot must accommodate single potential write-in. District overlaps with another county so reporting by district (with county tabulators) only gives totals for this county.
- State House District #3—One candidate running unopposed, registered as an Independent. Ballot must accommodate single potential write-in. District completely within the county.

In each of the above contests, if the voter requests a write-in candidate for a specific contest, the voting system will assist the voter to create the write-in choice. The voting system makes no attempt to validate write-in candidates for eligibility, but may appropriately restrict the number of write-ins allowed for a given contest. Validation of write-ins will be accomplished by a separate mechanism.

6.2.3 Local County Council contest

There are 10 candidates who have satisfied the requirements to be listed on the County Council ballot. Seven of the candidates are registered with one of the 4 political parties active in the county, one is a registered independent and two are not registered with any political party. The two not registered candidates are NOT registered Independents, so the ballot position allocated for political party affiliation for those two candidates must be left blank.

Each voter is allowed to vote for 4 candidates and may, if they wish, add up to 4 write-in candidates. The 4 candidates with the 4 highest vote totals are the winners. As with the single winner contests, there is no provision in local election law for handling ties!

6.2.4 Local ballot initiatives

There are four ballot initiatives with Yes or No votes possible, but not all initiatives are county wide! The first two initiatives are county-wide and apply to every precinct. The third initiative applies only to precincts in State House District #2 (the one that overlaps with another county). The fourth initiative applies only to two precincts that are their own municipality. The municipality crosses between State House districts 1 and 2.

6.2.5 Municipal council contest

There is a single municipality in the county that is holding a City Council contest with 5 candidates who have satisfied the requirements to be listed on the ballot. The selection process to appear on the ballot was done in a way that precludes write-ins, so these five candidates are the only possible candidates for two new City Council positions. In addition, this is a non-political contest, so candidate affiliations are not modelled or shown on the ballot; however, candidates may be endorsed, or not, by the political parties. Each voter is allowed two votes, but the cumulative voting variation is used, meaning that both votes could be cast for the same candidate. The two candidates with the two highest vote totals are the winners. As with the other elections above, there is no provision in local election law for handling ties.

6.2.6 Political parties

The county requires that the voting system keep track of all political parties for which a legal candidate for office has a declared affiliation and for which there is a provision in election law to carry that affiliation on the ballot. From the above it can be determined that there are five political parties that must be represented in any voting system used by the county. We assume that the five political parties have unique identifiers and unique names as shown in Table 14.

Table 14: Party information

PartyID	Name	FormalName
PP_NAP	Action	National Action Party
PP_SCP	Conservation	State Conservation Party
PP_SRI	Independent	State Registered Independent
PP_NMP	Moderate	National Moderate Party
PP_NPP	Progressive	National Progressive Party

Local law provides that the Name of the recognized political party is the name that will be placed on the ballot when required by election law. The state Registered Independent Party is treated slightly differently by state law in that any person may register as a State Independent, any candidate may have that affiliation appear for them on a ballot, but state law prohibits Registered Independents from forming any organization that endorses candidates or takes positions on issues.

The National Moderate Party and National Progressive Party are the two political parties that have an affiliated candidate for each national and statewide contest. The National Action Party is the affiliation of the third party candidate in the Presidential contest and the State Conservation Party is the affiliation of the third party candidate in the U.S. House and State Senate seats. Two State House candidates are registered Independents.

Three County Council candidates have Progressive affiliation, two have Moderate affiliation, two have Conservation affiliation, one is a registered Independent, and two have no affiliation. Although it is theoretically possible for a candidate to be affiliated with more than one political party, that phenomenon is so rare as to not be captured in the Votetest abstract model. If a candidate is a member of more than one party, then the candidate will have to choose which single affiliation gets printed on the ballot.

The Moderate, Progressive and Conservation parties have each endorsed a relatively full slate of candidates, including four County Council candidates. Since not every party has 4 County Council candidates, they may sometimes endorse a candidate from a rival party. The National Action party has only endorsed a partial set of candidates, but for County Council has endorsed the Independent candidate, the two Conservation candidates, and one non-affiliated candidate. These endorsements will not be visible on the ballot, which does not include straight-party voting, but the county would like to record them in the election definition anyway. The Moderate and Progressive parties have endorsed positions on the countywide ballot initiatives but not on local ballot initiatives. The Conservation party has endorsed candidates and positions on all of the local contests and local ballot initiatives.

6.2.7 Precincts

There are ten election precincts in the county distributed across the State House districts, with four in H1, three in H2 and three in H3. Two of the precincts make up the municipality that crosses districts 1 and 2. Table 15 gives the relationships among the precincts, the single State Senate district, the three State House districts, the single municipality, and the four ballot initiatives.

Table 15: Precinct relationships

PrecinctID	SS_District	SH_District	Municipality	BallotInitiatives
P01	S1	H1		B1, B2
P02	S1	H1		B1, B2
P03	S1	H1		B1, B2
P04	S1	H1	M1	B1, B2, B4
P05	S1	H2	M1	B1, B2, B3, B4
P06	S1	H2		B1, B2, B3
P07	S1	H2		B1, B2, B3
P08	S1	H3		B1, B2
P09	S1	H3		B1, B2
P10	S1	H3		B1, B2

6.2.8 Reporting requirements

State law requires that election results be reported by State Senate districts, State House districts, counties, and precincts. Local law requires that results for all municipalities be reported separately, so the single municipality that crosses State House districts 1 and 2 must be reported separately. Election law also requires election districts to maintain statistics by ballot category, e.g., Absentee, Provisional, Challenged, etc.

6.3 Modelling the election in Votetest

The first step in using Votetest to create test cases for this county election is to model the candidates and county requirements for this election as presented in Section 6.2. Beginning with the Votetest abstract model presented in Section 3.4, this section looks at subsets of that model to capture the above example information.

6.3.1 Ballot styles

Table 15 indicates that there will need to be at least five different ballot styles in order to capture the different ballot requirements for the different contests in each of the precincts; one style for precincts P1 through P3, a second and third styles for the unique requirements of precincts P4 and P5, a fourth style for precincts P6 and P7, and a fifth style for precincts P8 through P10. Some election districts may choose to have as many ballot styles as they have precincts, reasoning that the precinct number may be printed on the ballot, thereby giving each precinct a unique ballot style.

However, the election board of this county chooses to model only the five ballot styles minimally necessary to represent all the contests of this election.

The model subset shown in Figure 10 captures the required information for a ballot style. The BallotStyle class represents the structure of an unvoted ballot. In this example, the BallotStyle class will have five instances, each with a unique name. The county election board decides to identify the ballot styles and give them unique names as shown in Table 16.

Figure 10: Model subset for ballot styles

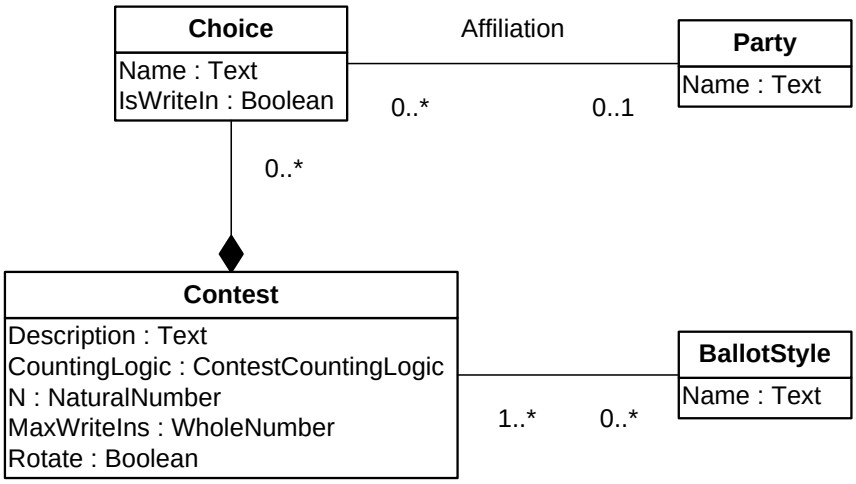


Table 16: Ballot styles

Local Id	Name	Description
S1	H1B1B2	House District #1 countywide initiatives only
S2	H1B1B2M1B4	House District #1 countywide and municipality contests
S3	H2B1B2B3	House District #2 countywide and district initiatives
S4	H2B1B2B3M1B4	House District #2 countywide, district and municipality contests
S5	H3B1B2	House District #3 countywide initiatives only

Unique identifiers for BallotStyles are implicit in the abstract Votetest model, via object identity; the concrete database schema uses explicit integer identifiers to achieve the same effect.

The many-to-many association between BallotStyle and Contest indicates that a contest may exist without being assigned to a ballot style, a ballot style consists of one or more contests, and a contest may appear on multiple different ballot styles.

The Contest class represents each of the contests described in Section 6.2. In total there will be 13 contest instances: one for President, one for U.S. Senate, one for U.S. House, one for State Senate, three for State House, one for County Council, one for municipal City Council, and four for ballot initiatives. Ballot style S1 will be associated with 8 contests: one for President, one for U.S. Senate, one for U.S. House, one for State Senate, one for State House, one for County Council, and two for ballot initiatives. Similarly, ballot style S2 will be associated with 10 contests, S3 with 9 contests, S4 with 11 contests, and S5 with 8 contests.

The Choice class represents the voting alternatives for each contest. The single winner candidate contests (e.g., President, Senate, House) will each have from 1 to 3 choices: one choice for the unopposed candidate in State House District #3 and two or three choices for each of the other single winner candidate contests. Note that the model allows a contest to have 0 choices; this is possible in the situation where a contest exists, no candidates have satisfied the requirements to be placed on the ballot, but write-in votes are allowed. Each write-in vote may result in the creation of a new choice instance, with the IsWriteIn attribute set to *true*, but that possibility is not part of the original ballot definition.

The County Council contest will have 10 choices, one for each candidate whose name appears on the ballot. Similarly, the municipal City Council contest will have 5 choices. The four ballot initiative contests will each have two choices, one for a Yes vote and one for a No vote.

The association between Contest and Choice is one-to-many; the solid diamond represents that each choice instance is tightly bound to its parent contest. It is not possible for a choice to exist apart from its parent class or in more than one Contest. If the same person is a candidate in more than one contest, then two separate Choices exist, i.e., one for each Contest.

The many-to-one Affiliation association between Choice and Party maintains an optional affiliation for each candidate choice. A candidate may or may not have a party affiliation; if the affiliation exists and local law allows the affiliation to be printed on the ballot, it will be printed next to the candidate's name. Note that a Party instance may exist even if no candidate is affiliated with that party. Also note that in real life a candidate may have multiple party affiliations, but the model restricts the affiliation to at most one because there is no known election board that allows multiple affiliations to be listed on a ballot. Non-candidate choices (e.g., ballot initiatives) would normally not have any affiliation associations; this could be enforced, if desired, by a separate constraint. However, ballot initiatives could have endorsements by political parties (c.f. Section 6.3.4).

6.3.2 Contest attributes

The Contest class has five significant attributes. In addition, each instance of the class will have an instance identifier separate from the five attributes. The Description attribute identifies the contest, e.g., U.S. President, State Senate District #5, State House District #3, County Council, Municipal City Council, or Ballot Initiative #2. For ballot initiatives the Description may also include a summary of the question being voted.

The CountingLogic attribute consists of one of the four enumeration values indicated in Figure 8 for ContestCountingLogic. All of the contests except municipal City Council in this example are N-of-M; they may be specifically 1-of-1, 1-of-2, 1-of-3, or 4-of-10, but all are identified as N-of-M by this attribute. The municipal City Council contest will have CountingLogic set to Cumulative, since the candidates may receive multiple votes on each ballot. The other two counting logics are not part of this example, but are defined in the VVSG [1].

For N-of-M and Cumulative contests, N identifies the number of votes that the voter may allocate without overvoting. Typically this is also the number of winners in that contest, but not necessarily. (For N-of-M and Cumulative contests, the voting system only needs to gather votes and report the totals; the picking of winners may be an external process impacted by election law, late-breaking judicial rulings, etc.)

In this example, N will be set to 1 for all of the single-winner contests and all of the ballot initiatives, to 4 for the County Council contest, and to 2 for the municipal City Council contest.

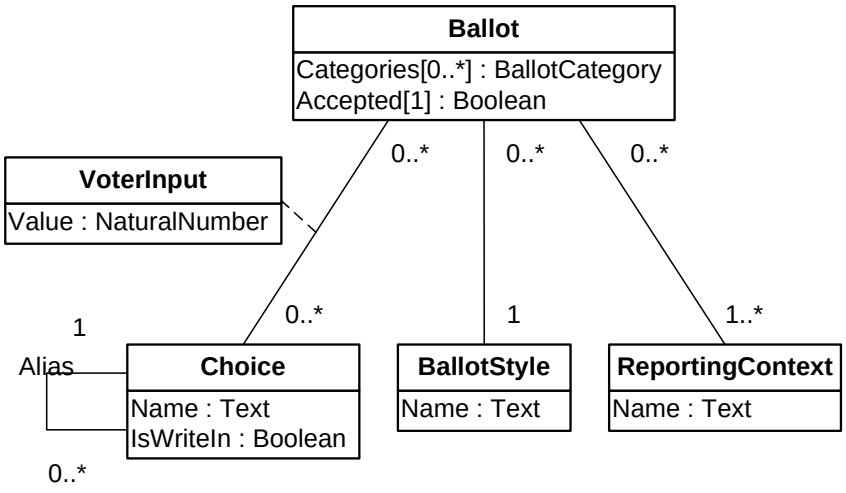
The MaxWriteIns attribute determines the provision that must be provided for write-ins by a voting system. In this example, all candidate contests will have this attribute set to 0, 1, 2, or 4, depending on whether or not write-ins are allowed for that contest. The candidate contests where write-ins are not allowed and all four of the ballot initiatives will have this attribute set to 0. The County Council contest will have it set to 4, the municipal City Council contest will have it set to 2, and all of the remaining candidate contests will have it set to 1.

The Rotate attribute determines whether or not the choices for a contest will have their positions on the ballot rotated during printing or display of the actual ballot instances. Note that ballot instances are represented in the model by the Ballot class (not yet discussed). This attribute is not relevant in the logic model of the VVSG, so for this example it could be set to *true* or *false* with no effect on the reporting requirements of the model.

6.3.3 Voted ballots

The Votetest abstract model makes a distinction between ballot structure and a voted ballot. Ballot structure is represented by the BallotStyle class whereas a voted ballot is represented by the Ballot class. Each Ballot instance is tied to exactly one ballot style and to one or more reporting contexts. The model subset in Figure 11 shows the attributes of a voted ballot and its potential relationships with other items.

Figure 11: Model subset for ballots



A Ballot instance is associated with exactly one ballot style. The ballot style determines the contests and the choices that will be presented to the voter. For each contest on the ballot, the voter may select one or more of the canonical choices, create a new write-in choice if that is permitted, or not vote in that contest. A voter is not required to make any choices and could turn in an unmarked ballot. The VoterInput class is included in the abstract model to be able to handle cumulative and ranked order voting. In this example, for all contests except municipal City Council, for each choice selected on a ballot, the Value attribute on VoterInput for a given association between Ballot and Choice carries Value = 1. In the municipal City Council contest, since a voter may cast up

to two votes for the same candidate, for each choice selected on the ballot, the Value attribute on VoterInput for a given association between Ballot and Choice could be set to either 1 or 2.

A Ballot instance is required to be associated with at least one reporting context. This is to guarantee that every ballot is accounted for and reported in at least one context. In our example, Ballot instances are created by the voting systems in each precinct, and precinct is the smallest reporting unit, so there will be 10 precinct instances for the ReportingContext class and each ballot will be associated with the relevant precinct. A ballot may also be indirectly associated with different reporting contexts through its parent ballot style. Other reporting contexts, both direct and indirect, are presented in Section 6.3.5.

A Ballot instance may be tagged with zero or more ballot categories chosen from the tags appearing in the BallotCategory enumeration given in Figure 8. The ballot category tags are not necessarily mutually exclusive or collectively exhaustive; new tags may be added to the list to support future acceptance or reporting rules. The purpose of the ballot category tags is to clearly distinguish those ballots that may be handled or reported according to a different set of rules. In our example, a regular ballot will have an empty BallotCategory attribute, an absentee ballot will carry an Absentee tag, and a provisional ballot will carry a Provisional tag. The conditions under which categories can legally be added or deleted from a ballot are specified by election law; controls on adding or deleting categories may be implemented by the voting system or they may be implemented procedurally. Either way, the voting system must be able to produce reports that properly reflect the categories that a ballot carries at the time the report is produced. In this section, only Absentee and Provisional categories will be used.

The Accepted attribute on a ballot instance indicates whether or not a ballot is to be counted. If Accepted is *true*, as it normally is for ballots having no unusual issues, then the ballot will be counted. If Accepted is *false*, then the ballot may be read and accounted for by the system but no choices for any contest will be counted. The Accepted attribute may be set by the voting system according to rules embedded in the voting system itself, or this attribute may be changed later by election officials after an analysis of provisional or challenged ballots. A voting system is generally not responsible for the current state of the Accepted attribute for provisional and challenged ballots; it must simply be able to produce reports that count, or not, all read ballots, and produce accurate reports depending on how this attribute is set at the time of reporting.

The Alias association on the Choice class is mostly used for write-in reconciliation. In this example, the voting system makes no attempt to interpret write-ins on the fly: Whenever a voter writes in a choice, the system invariably creates a new Choice instance with IsWriteIn = *true*. Later, through a separate process with humans in the loop, write-ins are "reconciled" to determine how their votes are to be credited. Each time a write-in is found to refer to a previously defined Choice, an instance of the Alias relationship is created to credit that write-in vote to the previously defined Choice. If a write-in is found to be completely new, no alias is created for it, and it will be reported as a separate choice.

In this example, without Alias relationships, every single write-in would be counted and reported as a distinct choice. However, this is only one valid approach. In a different jurisdiction with different equipment, policies and procedures, it would be perfectly valid, e.g., for an electronic voting system to create only one Choice instance for each distinct string of characters that some voter has "written in" and to refer to that same Choice instance the second and subsequent times that that string of characters appears. This reduces the amount of reconciliation that must be done later: Every occurrence of the same string of characters will be treated identically by the voting system. If two

different spellings of a candidate's name were found to be equivalent, only one instance of the Alias relationship would be needed to credit all of the affected votes appropriately.

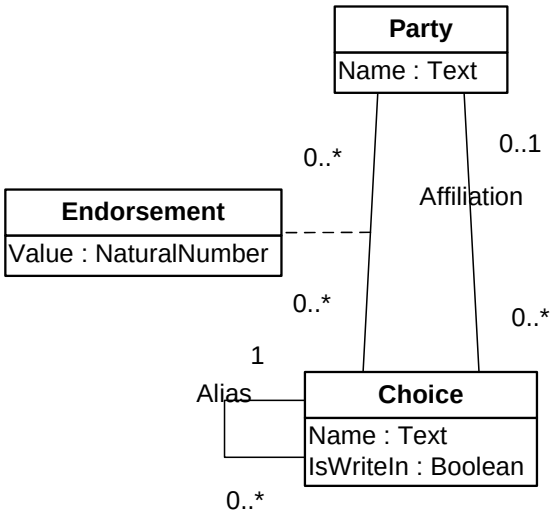
Rarely, aliases may be used to merge the counts for non-write-in choices. For example, if election law forces there to be separate choices on the ballot for each political party that endorsed a given candidate, these may be aliased in order to report a single, consolidated total for that candidate.

6.3.4 Provisions for endorsements

The Endorsement class in the Votetest model exists for the purpose of modelling elections that allow straight-party voting. This example election does not include straight-party selections; however, for completeness, the election board has decided to record the endorsements noted in Section 6.2.6.

The model subset shown in Figure 12 represents the independent concepts of party affiliation and party endorsement.

Figure 12: Model subset for affiliations and endorsements



Party and Choice classes and the Affiliation association were presented and discussed in Section 6.3.1. The Value attribute in the Endorsement class is similar to the Value attribute in the VoterInput class discussed in the previous section, but instead of specifying the vote of a particular voter for a particular choice, it specifies the vote that a particular party *recommends* that voters make.

For this example, there is a contest that uses Cumulative counting logic, but only the Conservation party has decided to weight its endorsement for that contest. All of the other party endorsements are simple endorsements of choices without weighting. Thus the Value attribute will be set to 1 for all associations between Party and Choice, except for the single association between the Conservation party and its weighted choice for municipal City Council. That association will have Value = 2.

Each of the four local parties have endorsed a full slate of candidates for County Council, have endorsed candidates with their own affiliation when that affiliation is known, and have endorsed a Yes or No vote on the two county-wide ballot initiatives. None of the mainstream parties has taken a

position on the non-county-wide ballot initiatives or has endorsed any of the municipal City Council candidates; however, the Conservation party did recommend a vote on all 4 ballot initiatives and endorsed a single candidate with double weight for the municipal City Council contest.

Conceptually, focusing only on the two council contests and the ballot initiatives, these endorsements might look like Table 17.

Table 17: Endorsements

Party	Contest	Choice (Affiliation)	Value
Action	County Council	Candidate #8 (Independent)	1
Action	County Council	Candidate #4 (Conservation)	1
Action	County Council	Candidate #9 (Conservation)	1
Action	County Council	Candidate #6 (empty)	1
Conservation	County Council	Candidate #4 (Conservation)	1
Conservation	County Council	Candidate #9 (Conservation)	1
Conservation	County Council	Candidate #7 (Moderate)	1
Conservation	County Council	Candidate #5 (Progressive)	1
Conservation	Initiative #1	Yes	1
Conservation	Initiative #2	No	1
Conservation	Initiative #3	Yes	1
Conservation	Initiative #4	No	1
Conservation	Municipal Council	Candidate #3 (Independent)	2
Moderate	County Council	Candidate #1 (Moderate)	1
Moderate	County Council	Candidate #9 (Conservation)	1
Moderate	County Council	Candidate #2 (Progressive)	1
Moderate	County Council	Candidate #7 (Moderate)	1
Moderate	Initiative #1	Yes	1
Moderate	Initiative #2	No	1
Progressive	County Council	Candidate #10 (Progressive)	1
Progressive	County Council	Candidate #5 (Progressive)	1
Progressive	County Council	Candidate #2 (Progressive)	1
Progressive	County Council	Candidate #7 (Moderate)	1
Progressive	Initiative #1	No	1
Progressive	Initiative #2	Yes	1

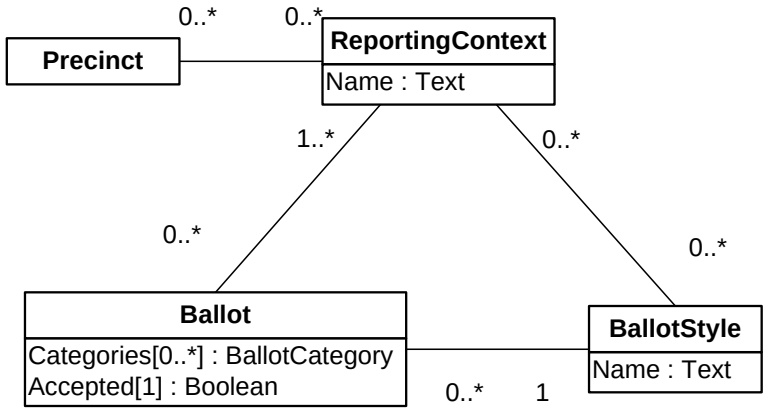
The physical representation of these endorsements as a normalized relational database table is given in Table 26. Note that the Contest column and the Affiliation in parentheses appearing in Table 17 are both superfluous since each Choice is linked to a unique Contest and to at most one Party; these convenience details go away in the physical representation.

6.3.5 Reporting of election results

As stated in Section 6.2.8, the county election board in this example is required to report results by U.S. congressional district, by State Senate and State House districts, by county, by municipality, by precinct, and report ballot counts for at least three ballot categories, i.e., Regular, Absentee and Provisional. Some additional reporting may be required later to account for other potential ballot

categories, but initially only these three are considered. Reporting by county will produce the same results as reporting by U.S. congressional district and reporting by State Senate district, so only county reporting is considered in this example. The model subset shown in Figure 13 represents the basic reporting requirements for this election.

Figure 13: Model subset for reporting



It's already established that for this example there are ten precincts, one election district, i.e., the county, and one tabulator for each precinct. All reporting requirements by election district, or by tabulators, are satisfied by including ReportingContext instances for one county (C1), 10 precincts (P01 through P10), three State House districts (H1, H2, H3), and one municipality (M1). The report generator automatically breaks down ballot counts by category in each reporting context, so there is no need to list the ballot categories as separate reporting contexts. The election board decides to consider the Regular ballot category as implicit, so the Categories attribute is left empty for every Regular ballot. The ReportingContext class will have 15 explicit instances, each with a unique Name.

The voting system will record explicit associations between Ballot and ReportingContext for each Precinct. As stated earlier, this ensures that every accepted ballot will be counted in its home precinct.

The remaining reporting contexts, i.e., County, one municipality and 3 State House districts, are associated with ballots only indirectly through BallotStyle. Table 18 records the explicit associations between ReportingContext and BallotStyle using the names of ballot styles from Section 6.3.1. In addition, the County reporting context (C1) will be linked to all five ballot styles.

Table 18: Ballot style—reporting context associations

ReportingContext	**BallotStyle**
StateHouseD1	H1B1B2
StateHouseD1	H1B1B2M1B4
StateHouseD2	H2B1B2B3
StateHouseD2	H2B1B2B3M1B4
StateHouseD3	H3B1B2
Municipality1	H1B1B2M1B4
Municipality1	H2B1B2B3M1B4

6.4 Representing the election in the database

Section 3.5 translates the classes and associations of the abstract model into tables in the concrete relational database representation. In general, classes map to relational tables with the same attributes, many-to-one associations are represented as a reference attribute in the table, and many-to-many associations map to tables with two columns, representing the source and target instances of the association. Some associations have a third attribute to capture the Value attribute from the VoterInput or Endorsement classes in the abstract model. If a class has an obvious unique attribute, e.g., unique Name, it may be used as the primary key of the table; otherwise, a table primary key is added with an integer data type.

The Ballot table is initially empty, and so are any association tables where a ballot is either the source or the target of the association. These tables are filled dynamically during the election as ballots are cast. The initial, non-empty tables, before voting, are shown in Table 19 through Table 26. Most tables contain a MyDescription attribute not present in the Votetest model and not used during test generation; it simply explains the attributes of the model in terms of the example.

Table 19: BallotStyle

Style Id	Name	MyDescription
1	H1B1B2	S1—House District #1 countywide initiatives only
2	H1B1B2M1B4	S2—House District #1 countywide and municipality initiatives
3	H2B1B2B3	S3—House District #2 countywide and district initiatives
4	H2B1B2B3M1B4	S4—House District #2 countywide, district and municipality initiatives
5	H3B1B2	S5—House District #3 countywide initiatives only

Table 20: Contest

ContestId	Description	CountingLogic	N	MaxWriteIns	Rotate
1	President	N-of-M	1	0	Yes
2	U.S. Senate	N-of-M	1	0	Yes
3	U.S. House	N-of-M	1	0	Yes
4	State Senate	N-of-M	1	0	Yes
5	State House #1	N-of-M	1	1	Yes
6	State House #2	N-of-M	1	1	Yes
7	State House #3	N-of-M	1	1	Yes
8	County Council	N-of-M	4	4	Yes
9	Ballot Initiative 1	N-of-M	1	0	No
10	Ballot Initiative 2	N-of-M	1	0	No
11	Ballot Initiative 3	N-of-M	1	0	No
12	Ballot Initiative 4	N-of-M	1	0	No
13	Muni City Council	Cumulative	2	0	No

Table 21: BallotStyleContestAssociation

StyleId	ContestId	MyDescription
1	1	H1B1B2—Pres
1	2	H1B1B2—US Senate
1	3	H1B1B2—US House
1	4	H1B1B2—State Senate
1	5	H1B1B2—State House
1	8	H1B1B2—Council
1	9	H1B1B2—initiative B1
1	10	H1B1B2—initiative B2
2	1	H1B1B2M1B4—Pres
2	2	H1B1B2M1B4—US Senate
2	3	H1B1B2M1B4—US House
2	4	H1B1B2M1B4—State Senate
2	5	H1B1B2M1B4—State House H1
2	8	H1B1B2M1B4—Council
2	9	H1B1B2M1B4—initiative B1
2	10	H1B1B2M1B4—initiative B2
2	12	H1B1B2M1B4—initiative B4
2	13	H1B1B2M1B4—City Council
3	1	H2B1B2B3—Pres
3	2	H2B1B2B3—US Senate
3	3	H2B1B2B3—US House
3	4	H2B1B2B3—State Senate
3	6	H2B1B2B3—State House H2
3	8	H2B1B2B3—Council
3	9	H2B1B2B3—initiative B1
3	10	H2B1B2B3—initiative B2
3	11	H2B1B2B3—initiative B3
4	1	H2B1B2B3M1B4—Pres
4	2	H2B1B2B3M1B4—US Senate
4	3	H2B1B2B3M1B4—US House
4	4	H2B1B2B3M1B4—State Senate
4	6	H2B1B2B3M1B4—State House H2
4	8	H2B1B2B3M1B4—Council
4	9	H2B1B2B3M1B4—initiative B1
4	10	H2B1B2B3M1B4—initiative B2
4	11	H2B1B2B3M1B4—initiative B3
4	12	H2B1B2B3M1B4—initiative B4
4	13	H2B1B2B3M1B4—City Council
5	1	H3B1B2—Pres
5	2	H3B1B2—US Senate
5	3	H3B1B2—US House
5	4	H3B1B2—State Senate
5	7	H3B1B2—State House H3

5	8	H3B1B2—Council
5	9	H3B1B2—initiative B1
5	10	H3B1B2—initiative B2

Table 22: Choice (before write-ins)

Choice Id	Contest Id	Name	Affiliation	Is WriteIn	MyDescription
1	1	PresidentC1	Moderate	No	US President Candidate 1
2	1	PresidentC2	Progressive	No	US President Candidate 2
3	1	PresidentC3	Action	No	US President Candidate 3
4	2	USSenateC1	Moderate	No	US Senate Candidate 1
5	2	USSenateC2	Progressive	No	US Senate Candidate 2
6	3	USHouseC1	Moderate	No	US House Candidate 1
7	3	USHouseC2	Progressive	No	US House Candidate 2
8	3	USHouseC3	Conservation	No	US House Candidate 3
9	4	SSenateC1	Moderate	No	State House Candidate 1
10	4	SSenateC2	Progressive	No	State House Candidate 2
11	4	SSenateC3	Conservation	No	State House Candidate 3
12	5	SHouseD1C1	Moderate	No	State House D1 Candidate 1
13	5	SHouseD1C2	Progressive	No	State House D1 Candidate 2
14	5	SHouseD1C3	Conservation	No	State House D1 Candidate 3
15	6	SHouseD2C1	Moderate	No	State House D2 Candidate 1
16	6	SHouseD2C2	Progressive	No	State House D2 Candidate 2
17	6	SHouseD2C3	Independent	No	State House D2 Candidate 3
18	7	SHouseD3C1	Independent	No	State House D3 Candidate 1
19	8	CCouncilC01	Moderate	No	Council Candidate 1
20	8	CCouncilC02	Progressive	No	Council Candidate 2
21	8	CCouncilC03		No	Council Candidate 3
22	8	CCouncilC04	Conservation	No	Council Candidate 4
23	8	CCouncilC05	Progressive	No	Council Candidate 5
24	8	CCouncilC06		No	Council Candidate 6
25	8	CCouncilC07	Moderate	No	Council Candidate 7
26	8	CCouncilC08	Independent	No	Council Candidate 8
27	8	CCouncilC09	Conservation	No	Council Candidate 9
28	8	CCouncilC10	Progressive	No	Council Candidate 10
29	9	Yes		No	Yes on Ballot Initiative 1
30	9	No		No	No on Ballot Initiative 1
31	10	Yes		No	Yes on Ballot Initiative 2
32	10	No		No	No on Ballot Initiative 2
33	11	Yes		No	Yes on Ballot Initiative 3
34	11	No		No	No on Ballot Initiative 3
35	12	Yes		No	Yes on Ballot Initiative 4
36	12	No		No	No on Ballot Initiative 4
37	13	MCCouncilC1		No	

38	13	MCCouncilC2		No	
39	13	MCCouncilC3		No	
40	13	MCCouncilC4		No	
41	13	MCCouncilC5		No	

Table 23: Party

Name	MyDescription
Action	National Action Party
Conservation	State Conservation Party
Independent	State Registered Independent
Moderate	National Moderate Party
Progressive	National Progressive Party

Table 24: ReportingContext

Name	MyDescription
County1	County
StateHouseD1	State House District #1
StateHouseD2	State House District #2
StateHouseD3	State House District #3
P01	Precinct 1
P02	Precinct 2
P03	Precinct 3
P04	Precinct 4
P05	Precinct 5
P06	Precinct 6
P07	Precinct 7
P08	Precinct 8
P09	Precinct 9
P10	Precinct 10
Municipality1	Municipality

Table 25: BallotStyleReportingContextAssociation

StyleId	ReportingContext	MyDescription
1	County1	County—S1
2	County1	County—S2
3	County1	County—S3
4	County1	County—S4
5	County1	County—S5
1	StateHouse1	House District #1—Not in municipality—S1
2	StateHouse1	House District #1—In municipality—S2
3	StateHouse2	House District #2—Not in municipality—S3

4	StateHouse2	House District #2—In municipality—S4
5	StateHouse3	House District #3—S5
2	Municipality1	Municipality—House District 1—S2
4	Municipality1	Municipality—House District 2—S4

Table 26: Endorsement

Party	ChoiceId	Value	MyDescription
Action	3	1	US President Candidate 3—Action
Action	4	1	US Senate Candidate 1—Moderate
Action	7	1	US House Candidate 2—Progressive
Action	11	1	State Senate Candidate 3—Conservation
Action	22	1	Council Candidate 4—Conservation
Action	24	1	Council Candidate 6
Action	26	1	Council Candidate 8—Independent
Action	27	1	Council Candidate 9—Conservation
Action	30	1	No on Ballot Initiative 1
Action	31	1	Yes on Ballot Initiative 2
Conservation	8	1	US House Candidate 3—Conservation
Conservation	11	1	State Senate Candidate 3—Conservation
Conservation	14	1	State House D1 Candidate 3—Conservation
Conservation	17	1	State House D2 Candidate 3—Independent
Conservation	22	1	Council Candidate 4—Conservation
Conservation	23	1	Council Candidate 5—Progressive
Conservation	25	1	Council Candidate 7—Moderate
Conservation	27	1	Council Candidate 9—Conservation
Conservation	29	1	Yes on Ballot Initiative 1
Conservation	32	1	No on Ballot Initiative 2
Conservation	33	1	Yes on Ballot Initiative 3
Conservation	36	1	No on Ballot Initiative 4
Conservation	39	2	Municipal Council Candidate 3
Moderate	1	1	US President Candidate 1—Moderate
Moderate	4	1	US Senate Candidate 1—Moderate
Moderate	6	1	US House Candidate 1—Moderate
Moderate	9	1	State Senate Candidate 1—Moderate
Moderate	12	1	State House D1 Candidate 1—Moderate
Moderate	15	1	State House D2 Candidate 1—Moderate
Moderate	18	1	State House D3 Candidate 1—Independent
Moderate	19	1	Council Candidate 1—Moderate
Moderate	20	1	Council Candidate 2—Progressive
Moderate	25	1	Council Candidate 7—Moderate
Moderate	27	1	Council Candidate 9—Conservation
Moderate	30	1	No on Ballot Initiative 1
Moderate	31	1	Yes on Ballot Initiative 2
Progressive	2	1	US President Candidate 2—Progressive

Progressive	5	1	US Senate Candidate 2—Progressive
Progressive	7	1	US House Candidate 2—Progressive
Progressive	10	1	State Senate Candidate 2—Progressive
Progressive	13	1	State House D1 Candidate 2—Progressive
Progressive	16	1	State House D2 Candidate 2—Progressive
Progressive	18	1	State House D3 Candidate 1—Independent
Progressive	20	1	Council Candidate 2—Progressive
Progressive	23	1	Council Candidate 5—Progressive
Progressive	25	1	Council Candidate 7—Moderate
Progressive	28	1	Council Candidate 10—Progressive
Progressive	29	1	Yes on Ballot Initiative 1
Progressive	32	1	No on Ballot Initiative 2

6.5 Generating relevant test cases

The purpose of this subsection is to describe how to use the tables identified in Section 6.4, together with the Votetest model, to produce test cases relevant to the characteristics of the election example presented. The tester may use tests already generated by Votetest to test the general validity of a voting system. Additional tests generated here serve to exercise the voting system in an environment that most closely approximates a specific election with specific ballot styles and reporting requirements. The following subsections describe the steps taken to assign relevant probability distributions and to create a representative number of test ballots. In general, a tester would follow these same steps to generate test ballots for a specific election.

6.5.1 Populate the tables

The first step is to use local election characteristics to prepare content for all of the non-empty tables identified in Section 6.4. The MyDescription attribute may be omitted as it is not used by Votetest. In addition, the Endorsement table is optional because in this election the straight-party voting variation is not used.

6.5.2 Assign Ballot number and general Ballot distributions

Table 27 indicates the distribution of sample ballots over the Categories and Accepted attributes of the Ballot class (presented in Section 6.3.3) that the tester in this example wants to generate. A probability is given for each category in the BallotCategory table; in addition, the probability of a ballot being rejected or blank is included. The total number of ballots that the tester wants to generate is 2000.

Each of the named probability distributions is treated as an independent, random binomial event. So we expect approximately 1 % of the 2000 generated ballots to have the Accepted attribute set to *false*, 1 % to be a blank ballot, 13 % to have Absentee in the Categories attribute, and 6 % to have Provisional in the Categories attribute. None of the other category entries in this table will occur at all.

Table 27: BallotCategory, rejected and blank ballot distributions

Name	Distribution
RejectedBallots	1 %
BlankBallots	1 %
Absentee	13 %
Challenged	0 %
Early	0 %
IneligibleVoter	0 %
InPerson	0 %
NotRegistered	0 %
Provisional	6 %
Regular	0 %
WrongPrecinct	0 %

Note that all four of these potential events are treated as independent. Thus it is possible for a Provisional ballot to be an Absentee ballot. It is also possible that a Rejected ballot be Provisional, Absentee, Blank, or any combination of the three. If desired, a testing facility could set up a more sophisticated ballot distribution with conditional probabilities for those events that have interdependencies.

The Ballot table is created first with 2000 rows. Then the Accepted attribute is set to *true* or *false* depending on the RejectedBallots distribution of 1 %. Next the BallotCategoryAssociation table is populated—first with 13 % Absentee associations and then, independently, with 6 % Provisional distributions. Then 1 % of the ballots are independently tagged as blank ballots so that no associations will ever be created from a blank ballot to a Choice instance.

6.5.3 Assign Precinct and BallotStyle distributions

Most elections will have pre-determined relationships among the precincts, reporting contexts, and ballot styles. Using the precinct relationships information in Section 6.2.7, Table 28 shows the probability distributions for assigning ballots to precincts. The other distributions are then derived from the precinct distribution.

The general abstract model in Section 3.4 supports precincts that may not be the smallest reporting unit. If that is the case, then a testing facility could assign probability distributions to the smallest reporting units and then have those probabilities distributed over all of the other reporting units and ballot styles.

The precinct probability distributions should add up to 100 %. The assignment of ballots to precincts is not considered to be a collection of independent events. Instead, the assignment of precincts is handled as a single, random multinomial event. Using the precinct probability distributions given in Table 28 one is able to populate the BallotReportingContextAssociation table between Ballot and ReportingContext to satisfy the requirement that each ballot be assigned to at least one reporting context, i.e., its precinct.

Table 28: Precinct and BallotStyle distributions

Precinct	P_Dist	District	D_Dist	Munic	M_Dist	BallotStyle	BS_Dist
P01	5 %	H1	28 %			H1B1B2	24 %
P02	12 %	H1	28 %			H1B1B2	24 %
P03	7 %	H1	28 %			H1B1B2	24 %
P04	4 %	H1	28 %	M1	12 %	H1B1B2M1B4	4 %
P05	8 %	H2	36 %	M1	12 %	H2B1B2B3M1B4	8 %
P06	13 %	H2	36 %			H2B1B2B3	28 %
P07	15 %	H2	36 %			H2B1B2B3	28 %
P08	6 %	H3	36 %			H3B1B2	36 %
P09	11 %	H3	36 %			H3B1B2	36 %
P10	19 %	H3	36 %			H3B1B2	36 %

6.5.4 Assign undervote distributions

The Contest table contains a list of potential contests for each ballot. In practice, some ballots may leave a given contest unvoted. It is possible to add one additional column to the Contest table to assign a probability that the contest will be left unvoted on a given ballot. Contests like a national presidential contest may have a very low probability of being left unvoted, whereas some complex ballot initiatives may have a relatively high probability of being left unvoted. One could assume different probabilities for each contest, depending on which ballot style is being used, or in which political unit a ballot is cast. For simplicity, this section assumes that all ballots will have the same probability of leaving a specific contest unvoted.

Table 29 carries the undervote probability distributions for each contest. The undervote distribution is independent of the CountingLogic, N, MaxWriteIns, or Rotate attributes of a contest, so those columns are elided.

Table 29: Undervote distributions

ContestId	Description	Unvoted
1	President	1 %
2	U.S. Senate	2 %
3	U.S. House	3 %
4	State Senate	4 %
5	State House #1	5 %
6	State House #2	5 %
7	State House #3	22 %
8	County Council	3 %
9	Ballot Initiative 1	15 %
10	Ballot Initiative 2	15 %
11	Ballot Initiative 3	20 %
12	Ballot Initiative 4	25 %
13	Muni City Council	30 %

Each time a contest appears on a ballot, whether or not it is left unvoted is treated as a random,

independent binomial event with the above probabilities. Each ballot is linked to a ballot style, and each ballot style is linked to a set of contests. Thus a join of these tables, with the elimination of blank ballots, produces a link between a specific ballot and the contests relevant to that ballot. Each row of that table is considered as an independent event for being voted or unvoted and rows are tagged as appropriate. Ballots linked to an unvoted contest will not be linked to any choices for that contest.

6.5.5 Assign Choice distributions

The Choice table as initialized in Table 22 consists of a collection of choices for each contest. Since a number of candidates are already known to be running specifically as write-in candidates, choices for these candidates may also be created in advance of the election. The tester prepares a list of likely write-ins for each contest as shown in Table 30.

Table 30: Choices for anticipated write-ins

ChoiceId	ContestId	Name	Affiliation	IsWriteIn	MyDescription
1001	5	SHouseD1 Writein1		Yes	Canonical
1002	5	SHouseD1 Writein2		Yes	Canonical
1003	5	Invalid Writein		Yes	Indeterminate
1004	6	SHouseD2 Writein1		Yes	Canonical
1005	6	Invalid Writein		Yes	Indeterminate
1006	7	SHouseD3 Writein1		Yes	Canonical
1007	7	SHouseD3 Writein2		Yes	Canonical
1008	7	SHouseD3 Writein3		Yes	Canonical
1009	7	Invalid Writein		Yes	Indeterminate
1010	8	CCouncil Writein1		Yes	Canonical
1011	8	CCouncil Writein2		Yes	Canonical
1012	8	CCouncil Writein3		Yes	Canonical
1013	8	CCouncil Writein4		Yes	Canonical
1014	8	CCouncil Writein5		Yes	Canonical
1015	8	Invalid Writein		Yes	Indeterminate

None of the choices listed in Table 30 will ever be referenced directly by a given ballot. Instead, each time a write-in is chosen on a ballot, the new Choice created for it will be aliased to one of these "canonical" write-ins, or to one of the invalid write-ins if the given write-in is indeterminate or illegal. If a valid write-in were received that did not match anything on this list, then the Choice instance created for it would not be aliased to anything. See Section 6.3.3 for more discussion on the use of aliasing in this example.

The tester now needs to assign a probability distribution to each of the possible choices, including potential write-ins. The result is shown in Table 31.

Table 31: Canonical choice and write-in distribution

ChoiceId	ContestId	Name	Affiliation	IsWriteIn	MyDescription	Dist
1	1	PresidentC1	Moderate	No	US President Cand 1	41 %
2	1	PresidentC2	Progressive	No	US President Cand 2	41 %
3	1	PresidentC3	Action	No	US President Cand 3	18 %
4	2	USSenateC1	Moderate	No	US Senate Cand 1	50 %
5	2	USSenateC2	Progressive	No	US Senate Cand 2	50 %
6	3	USHouseC1	Moderate	No	US House Cand 1	33 %
7	3	USHouseC2	Progressive	No	US House Cand 2	34 %
8	3	USHouseC3	Conservation	No	US House Cand 3	33 %
9	4	SSenateC1	Moderate	No	State House Cand 1	34 %
10	4	SSenateC2	Progressive	No	State House Cand 2	33 %
11	4	SSenateC3	Conservation	No	State House Cand 3	33 %
12	5	SHouseD1C1	Moderate	No	State House D1 Cand 1	36 %
13	5	SHouseD1C2	Progressive	No	State House D1 Cand 2	36 %
14	5	SHouseD1C3	Conservation	No	State House D1 Cand 3	21 %
1001	5	SHouseD1 Writein1		Yes	Canonical	3 %
1002	5	SHouseD1 Writein2		Yes	Canonical	2 %
1003	5	Invalid Writein		Yes	Indeterminate	2 %
15	6	SHouseD2C1	Moderate	No	State House D2 Cand 1	42 %
16	6	SHouseD2C2	Progressive	No	State House D2 Cand 2	42 %
17	6	SHouseD2C3	Independent	No	State House D2 Cand 3	10 %
1004	6	SHouseD2 Writein1		Yes	Canonical	5 %
1005	6	Invalid Writein		Yes	Indeterminate	1 %
18	7	SHouseD3C1	Independent	No	State House D3 Cand 1	40 %
1006	7	SHouseD3 Writein1		Yes	Canonical	45 %
1007	7	SHouseD3 Writein2		Yes	Canonical	8 %
1008	7	SHouseD3 Writein3		Yes	Canonical	4 %
1009	7	Invalid Writein		Yes	Indeterminate	3 %
19	8	CCouncilC01	Moderate	No	Council Candidate 1	25 %
20	8	CCouncilC02	Progressive	No	Council Candidate 2	20 %
21	8	CCouncilC03		No	Council Candidate 3	45 %
22	8	CCouncilC04	Conservation	No	Council Candidate 4	15 %
23	8	CCouncilC05	Progressive	No	Council Candidate 5	30 %
24	8	CCouncilC06		No	Council Candidate 6	20 %
25	8	CCouncilC07	Moderate	No	Council Candidate 7	40 %
26	8	CCouncilC08	Independent	No	Council Candidate 8	20 %
27	8	CCouncilC09	Conservation	No	Council Candidate 9	20 %
28	8	CCouncilC10	Progressive	No	Council Candidate 10	25 %
1010	8	CCouncil Writein1		Yes	Canonical	28 %
1011	8	CCouncil Writein2		Yes	Canonical	15 %
1012	8	CCouncil Writein3		Yes	Canonical	10 %
1013	8	CCouncil Writein4		Yes	Canonical	10 %

1014	8	CCouncil Writein5		Yes	Canonical	5 %
1015	8	Invalid Writein		Yes	Indeterminate	3 %
29	9	Yes		No	Yes on Ballot Initiative 1	60 %
30	9	No		No	No on Ballot Initiative 1	40 %
31	10	Yes		No	Yes on Ballot Initiative 2	50 %
32	10	No		No	No on Ballot Initiative 2	50 %
33	11	Yes		No	Yes on Ballot Initiative 3	40 %
34	11	No		No	No on Ballot Initiative 3	60 %
35	12	Yes		No	Yes on Ballot Initiative 4	51 %
36	12	No		No	No on Ballot Initiative 4	49 %
37	13	MCCouncilC1		No		5 %
38	13	MCCouncilC2		No		35 %
39	13	MCCouncilC3		No		10 %
40	13	MCCouncilC4		No		20 %
41	13	MCCouncilC5		No		30 %

A test facility may use the above distributions in a variety of ways. For example, suppose a single-winner contest has 2 candidates each with a 50 % probability distribution. The tester could regard this contest as a single, random binomial event for each ballot, or it could regard the contest as two independent, random binomial events for each ballot. In the first case, each candidate would have a 50 % chance of being selected on the ballot and there would be no undervotes and no overvotes. In the second case there would be a 25 % chance of neither candidate receiving a vote, a 50 % chance of exactly one candidate receiving a vote, and a 25 % chance of both candidates receiving a vote. The second case would have 25 % undervotes and 25 % overvotes. The process would be different, but each candidate would receive, on average, the same number of votes.

For this example, we assume that each single-winner contest, i.e., each 1-of-M contest, whose choice probabilities add to 100 %, is treated as a single, random multinomial event with exactly one candidate receiving a vote on that ballot. There will be no overvotes for these contests. However, there may still be some undervotes because of the previous allocation for blank ballots and unvoted contests. This technique still works even if the choice probabilities add to less than 100 %; the difference between the sum and 100 % would just add to the undervote population.

For multiple-winner N-of-M contests, it is often more realistic to consider the contest to be a sequence of M independent, binomial events, with each candidate having its given probability of receiving a vote. We apply this technique to the County Council contest. Note that the above probabilities for choices in the County Council contest (ContestId = 8) sum to 331 %. This means that on average, a non-blank, voted ballot for this contest will select only 3.31 council candidates; however, there will be a distribution from 0 to 16 of the candidates being selected, likely resulting in a substantial number of undervotes and overvotes. One could modify the distribution to add or subtract from the sum, thereby adding to the number of undervoted or overvoted ballots.

For multiple winner N-of-M contests, it is also possible to consider the contest as a sequence of N independent multinomial events, with each candidate having its given probability of receiving a vote on each event. We apply this technique to the municipal City Council contest (ContestId = 13) where the counting logic is Cumulative, up to 2 votes allowed, and the 5 given probabilities add to exactly 100 %. Each candidate will have 2 independent chances to receive a vote, so some will receive 0 votes, some exactly 1 vote, and some exactly 2 votes. Since this contest has cumulative

counting logic, there will be no overvotes from this approach and no new undervotes. There will still be some undervotes because of the previous allocation for blank ballots and unvoted contests.

6.5.6 Generate a simulated election

It's now time to vote! We simulate an election based on the above probability distributions by taking the following actions:

1. Populate the Ballot table with 2000 ballots. Do not yet enforce the integrity constraint to BallotStyle.
2. Assign an appropriate value to the Accepted attribute, based upon the binomial distribution for RejectedBallots defined in Section 6.5.2.
3. Populate the BallotCategoryAssociation table according to the binomial distributions for Categories defined in Section 6.5.2.
4. Populate the BallotReportingContextAssociation table according to the Precinct probability distributions defined in Section 6.5.3.
5. Update the Ballot table to assign a value to the StyleId attribute using the Precincts just generated for each ballot to link to ReportingContext and the existing BallotStyleReportingContextAssociation table to identify the appropriate BallotStyle ID. Turn on the referential integrity check from Ballot to BallotStyle.
6. Populate the Choice table with new choices for the write-ins identified in Table 30.
7. Populate the Choice table with a new choice for each write-in in any contest where a write-in choice is generated by the distributions given in Table 31.
8. Populate the Alias table to have each write-in choice linked to the appropriate "canonical" write-in. Recall that the canonical write-ins in the Choice table do not identify a choice on any ballot; instead the dynamically generated write-in choices are now linked to a canonical write-in if one exists.
9. Populate the VoterInput table for all 1-of-M contests making sure not to add choices for ballots that have been tagged as blank ballots or for ballot-contest combinations that have been tagged as unvoted contests.
10. Populate the VoterInput table for the County Council contest with the same provisions for blank ballots and unvoted contests.
11. Populate the VoterInput table for the municipal City Council contest with the same provisions for blank ballots and unvoted contests.

A Microsoft Access database implementing this example appears in the TestGenerator subdirectory of the Votetest distribution with file name WorkedExampleDataGenerator.mdb. The database contains all of the relational database tables, plus macros that implement the steps of the test data generation process. The READ_ME base table contains additional documentation specific to the Access database. A tester could modify the data and rerun the macros to generate test data for example elections with different contests and probability distributions.

N.B., An alternative test generator was documented in Section 5.7 as part of the Votetest advanced test development environment.

6.5.7 Input ballots to a specific voting system

Use the simulated ballots generated in the previous subsection as input to a specific voting system under test. This may take some effort to define a suitable interface to the voting system, since no standard test interface is required by the VVSG.

Cast all of the ballots and then compare the results from the voting system to the reports generated by Votetest. They should be identical.

6.6 Reports generated by Votetest

The database tables generated in Section 6.5 were transferred to the advanced test development environment of Votetest (discussed in Section 5) by exporting them to a tab-delimited text format and then importing them with the PostgreSQL COPY command. The Votetest infrastructure for integrity checking and reporting was then invoked as for any other test case. The results are presented in the following subsections.

To conserve space, not all reports are fully presented below. However, the Votetest integrity check report, the county report, the three district reports, the municipality report, and representative examples of the precinct reports are presented in full.

6.6.1 Integrity checks

```
######################################################################
           BEGIN TEST CASE OUTPUT   2008-03-11 15:47:35-04
######################################################################

$Id: Documentation.tex 511 2009-06-22 19:16:58Z dflater $

----- Begin integrity check output -- All results should be empty -----

 Out Of Range Voter Inputs
 ballotid | choiceid | value
----------+----------+-------
(0 rows)

Out Of Range Endorsements
 party | choiceid | value
-------+----------+-------
(0 rows)

 Extraneous Inputs
 ballotid | choiceid | value
----------+----------+-------
(0 rows)

Unreported Ballots
 ballotid
```

```
----------
(0 rows)

 Double Votes
 ballotid | choiceid | count | sum
----------+----------+-------+-----
(0 rows)

 Cross Contest Aliases
 aliasid | aliasname | aliascontestid | choiceid | choicename | choicecontestid
---------+-----------+----------------+----------+------------+-----------------
(0 rows)

 Double Indirect Aliases
 aliasid | aliasedaliasid
---------+----------------
(0 rows)

Ballot Styles With More Than One Straight Party Contest
 styleid
---------
(0 rows)

Non-Existent Parties In Straight Party Contest
 contestid | name
-----------+------
(0 rows)

Circular Straight Party Endorsements
 party | choiceid
-------+----------
(0 rows)

 Endorsed Aliases
 party | choiceid | value
-------+----------+-------
(0 rows)

 Straight Party Overrides
 ballotid | contestid | choiceid | value
----------+-----------+----------+-------
(0 rows)

 Too Many Write-Ins
 ballotid | contestid | writeinscount
----------+-----------+---------------
(0 rows)
```

```
----- End integrity check output ---- All results should be empty -----

------------------------------------------------------------------------------

Report total volume: 155990
  - Includes optional reporting of blank ballots.
  - Excludes separate reporting of ballots cast vs. read.

##############################################################################
```

6.6.2 Countywide results

Report for context County1 generated 2008-03-11 15:48:07-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		2000	1979
	Absentee	254	251
	Provisional	120	119
	Blank	19	19
H1B1B2		479	475
	Absentee	68	67
	Provisional	30	30
	Blank	5	5
H1B1B2M1B4		87	87
	Absentee	11	11
	Provisional	6	6
	Blank	2	2
H2B1B2B3		547	540
	Absentee	61	61
	Provisional	33	32
	Blank	6	6
H2B1B2B3M1B4		148	146
	Absentee	21	20
	Provisional	7	7
	Blank	1	1
H3B1B2		739	731
	Absentee	93	92
	Provisional	44	44
	Blank	5	5

VOTE TOTALS

President	
PresidentC1 (Moderate)	796
PresidentC2 (Progressive)	778
PresidentC3 (Action)	370
Overvotes	0
Undervotes	35
Counted ballots	1979
Balance	0
U.S. Senate	
USSenateC2 (Progressive)	968
USSenateC1 (Moderate)	964
Overvotes	0
Undervotes	47
Counted ballots	1979
Balance	0
U.S. House	
USHouseC2 (Progressive)	678
USHouseC1 (Moderate)	625
USHouseC3 (Conservation)	616
Overvotes	0
Undervotes	60
Counted ballots	1979
Balance	0
State Senate	
SSenateC2 (Progressive)	647
SSenateC1 (Moderate)	627
SSenateC3 (Conservation)	620
Overvotes	0
Undervotes	85
Counted ballots	1979
Balance	0
State House #1	
SHouseD1C1 (Moderate)	200
SHouseD1C2 (Progressive)	193
SHouseD1C3 (Conservation)	108
SHouseD1 Writein1 (write-in)	12
SHouseD1 Writein2 (write-in)	11
Invalid Writein (write-in)	7
Overvotes	0
Undervotes	31

Counted ballots	562
Balance	0
State House #2	
SHouseD2C1 (Moderate)	269
SHouseD2C2 (Progressive)	266
SHouseD2C3 (Independent)	72
SHouseD2 Writein1 (write-in)	44
Invalid Writein (write-in)	2
Overvotes	0
Undervotes	33
Counted ballots	686
Balance	0
State House #3	
SHouseD3 Writein1 (write-in)	266
SHouseD3C1 (Independent)	221
SHouseD3 Writein2 (write-in)	41
SHouseD3 Writein3 (write-in)	20
Invalid Writein (write-in)	14
Overvotes	0
Undervotes	169
Counted ballots	731
Balance	0
County Council	
CCouncilC03	595
CCouncilC07 (Moderate)	511
CCouncilC05 (Progressive)	378
CCouncil Writein1 (write-in)	330
CCouncilC10 (Progressive)	329
CCouncilC01 (Moderate)	322
CCouncilC06	248
CCouncilC02 (Progressive)	241
CCouncilC08 (Independent)	233
CCouncilC09 (Conservation)	230
CCouncilC04 (Conservation)	176
CCouncil Writein2 (write-in)	165
CCouncil Writein3 (write-in)	104
CCouncil Writein4 (write-in)	102
CCouncil Writein5 (write-in)	62
Invalid Writein (write-in)	30
Overvotes	1584
Undervotes	2276
Counted ballots	1979
Balance	0
Ballot Initiative 1	

Yes	987
No	663
Overvotes	0
Undervotes	329
Counted ballots	1979
Balance	0
Ballot Initiative 2	
No	843
Yes	832
Overvotes	0
Undervotes	304
Counted ballots	1979
Balance	0
Ballot Initiative 3	
No	311
Yes	207
Overvotes	0
Undervotes	168
Counted ballots	686
Balance	0
Ballot Initiative 4	
Yes	93
No	86
Overvotes	0
Undervotes	54
Counted ballots	233
Balance	0
Muni City Council	
MCCouncilC2	115
MCCouncilC5	91
MCCouncilC4	65
MCCouncilC3	34
MCCouncilC1	13
Overvotes	0
Undervotes	148
Counted ballots	233
Balance	0

6.6.3 Precinct results

Report for context P01 generated 2008-03-11 15:48:07-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		93	93
	Absentee	19	19
	Provisional	6	6
	Blank	0	0
H1B1B2		93	93
	Absentee	19	19
	Provisional	6	6

VOTE TOTALS

President	
PresidentC2 (Progressive)	42
PresidentC1 (Moderate)	37
PresidentC3 (Action)	14
Overvotes	0
Undervotes	0
Counted ballots	93
Balance	0
U.S. Senate	
USSenateC1 (Moderate)	47
USSenateC2 (Progressive)	46
Overvotes	0
Undervotes	0
Counted ballots	93
Balance	0
U.S. House	
USHouseC2 (Progressive)	32
USHouseC1 (Moderate)	31
USHouseC3 (Conservation)	29
Overvotes	0
Undervotes	1
Counted ballots	93
Balance	0
State Senate	
SSenateC1 (Moderate)	35
SSenateC3 (Conservation)	31
SSenateC2 (Progressive)	27
Overvotes	0
Undervotes	0
Counted ballots	93

Balance	0
State House #1	
SHouseD1C1 (Moderate)	33
SHouseD1C2 (Progressive)	27
SHouseD1C3 (Conservation)	17
SHouseD1 Writein2 (write-in)	3
Invalid Writein (write-in)	3
SHouseD1 Writein1 (write-in)	1
Overvotes	0
Undervotes	9
Counted ballots	93
Balance	0
County Council	
CCouncilC03	27
CCouncilC05 (Progressive)	23
CCouncilC07 (Moderate)	17
CCouncilC02 (Progressive)	16
CCouncilC10 (Progressive)	14
CCouncil Writein1 (write-in)	13
CCouncilC01 (Moderate)	12
CCouncilC06	12
CCouncilC08 (Independent)	11
CCouncilC09 (Conservation)	11
CCouncil Writein2 (write-in)	9
CCouncil Writein4 (write-in)	7
CCouncilC04 (Conservation)	6
CCouncil Writein3 (write-in)	5
CCouncil Writein5 (write-in)	3
Invalid Writein (write-in)	1
Overvotes	80
Undervotes	105
Counted ballots	93
Balance	0
Ballot Initiative 1	
Yes	40
No	34
Overvotes	0
Undervotes	19
Counted ballots	93
Balance	0
Ballot Initiative 2	
Yes	46
No	36
Overvotes	0

Undervotes	11
Counted ballots	93
Balance	0

Report for context P04 generated 2008-03-11 15:48:08-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		87	87
	Absentee	11	11
	Provisional	6	6
	Blank	2	2
H1B1B2M1B4		87	87
	Absentee	11	11
	Provisional	6	6
	Blank	2	2

VOTE TOTALS

President	
PresidentC1 (Moderate)	29
PresidentC2 (Progressive)	29
PresidentC3 (Action)	22
Overvotes	0
Undervotes	7
Counted ballots	87
Balance	0
U.S. Senate	
USSenateC2 (Progressive)	44
USSenateC1 (Moderate)	40
Overvotes	0
Undervotes	3
Counted ballots	87
Balance	0
U.S. House	
USHouseC2 (Progressive)	37
USHouseC3 (Conservation)	25
USHouseC1 (Moderate)	20
Overvotes	0

Undervotes	5
Counted ballots	87
Balance	0

State Senate	
SSenateC2 (Progressive)	30
SSenateC3 (Conservation)	28
SSenateC1 (Moderate)	25
Overvotes	0
Undervotes	4
Counted ballots	87
Balance	0

State House #1	
SHouseD1C1 (Moderate)	31
SHouseD1C2 (Progressive)	26
SHouseD1C3 (Conservation)	20
SHouseD1 Writein1 (write-in)	3
SHouseD1 Writein2 (write-in)	3
Invalid Writein (write-in)	0
Overvotes	0
Undervotes	4
Counted ballots	87
Balance	0

County Council	
CCouncilC03	25
CCouncilC07 (Moderate)	23
CCouncilC05 (Progressive)	17
CCouncil Writein1 (write-in)	15
CCouncilC08 (Independent)	14
CCouncilC10 (Progressive)	13
CCouncilC01 (Moderate)	12
CCouncilC09 (Conservation)	12
CCouncilC06	8
CCouncilC02 (Progressive)	7
CCouncilC04 (Conservation)	7
CCouncil Writein2 (write-in)	4
CCouncil Writein3 (write-in)	4
CCouncil Writein4 (write-in)	4
CCouncil Writein5 (write-in)	3
Invalid Writein (write-in)	2
Overvotes	88
Undervotes	90
Counted ballots	87
Balance	0

Ballot Initiative 1

Yes	41
No	34
Overvotes	0
Undervotes	12
Counted ballots	87
Balance	0

Ballot Initiative 2	
Yes	43
No	30
Overvotes	0
Undervotes	14
Counted ballots	87
Balance	0

Ballot Initiative 4	
No	34
Yes	33
Overvotes	0
Undervotes	20
Counted ballots	87
Balance	0

Muni City Council	
MCCouncilC2	47
MCCouncilC5	30
MCCouncilC4	28
MCCouncilC3	10
MCCouncilC1	5
Overvotes	0
Undervotes	54
Counted ballots	87
Balance	0

Report for context P05 generated 2008-03-11 15:48:08-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		148	146
	Absentee	21	20
	Provisional	7	7
	Blank	1	1

H2B1B2B3M1B4		148	146
	Absentee	21	20
	Provisional	7	7
	Blank	1	1

VOTE TOTALS

President	
PresidentC2 (Progressive)	63
PresidentC1 (Moderate)	53
PresidentC3 (Action)	29
Overvotes	0
Undervotes	1
Counted ballots	146
Balance	0
U.S. Senate	
USSenateC1 (Moderate)	70
USSenateC2 (Progressive)	69
Overvotes	0
Undervotes	7
Counted ballots	146
Balance	0
U.S. House	
USHouseC2 (Progressive)	53
USHouseC1 (Moderate)	46
USHouseC3 (Conservation)	42
Overvotes	0
Undervotes	5
Counted ballots	146
Balance	0
State Senate	
SSenateC3 (Conservation)	51
SSenateC1 (Moderate)	48
SSenateC2 (Progressive)	42
Overvotes	0
Undervotes	5
Counted ballots	146
Balance	0
State House #2	
SHouseD2C1 (Moderate)	61
SHouseD2C2 (Progressive)	58
SHouseD2C3 (Independent)	11
SHouseD2 Writein1 (write-in)	10

Invalid Writein (write-in)	1
Overvotes	0
Undervotes	5
Counted ballots	146
Balance	0
County Council	
CCouncilC07 (Moderate)	44
CCouncilC03	39
CCouncilC05 (Progressive)	34
CCouncilC09 (Conservation)	25
CCouncilC10 (Progressive)	24
CCouncil Writein1 (write-in)	22
CCouncilC01 (Moderate)	21
CCouncilC02 (Progressive)	19
CCouncilC04 (Conservation)	16
CCouncil Writein2 (write-in)	16
CCouncilC08 (Independent)	15
CCouncilC06	14
CCouncil Writein4 (write-in)	8
CCouncil Writein3 (write-in)	4
CCouncil Writein5 (write-in)	3
Invalid Writein (write-in)	3
Overvotes	120
Undervotes	157
Counted ballots	146
Balance	0
Ballot Initiative 1	
Yes	78
No	41
Overvotes	0
Undervotes	27
Counted ballots	146
Balance	0
Ballot Initiative 2	
No	66
Yes	60
Overvotes	0
Undervotes	20
Counted ballots	146
Balance	0
Ballot Initiative 3	
No	69
Yes	48
Overvotes	0

Undervotes	29
Counted ballots	146
Balance	0
Ballot Initiative 4	
Yes	60
No	52
Overvotes	0
Undervotes	34
Counted ballots	146
Balance	0
Muni City Council	
MCCouncilC2	68
MCCouncilC5	61
MCCouncilC4	37
MCCouncilC3	24
MCCouncilC1	8
Overvotes	0
Undervotes	94
Counted ballots	146
Balance	0

Report for context P08 generated 2008-03-11 15:48:08-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		129	129
	Absentee	17	17
	Provisional	5	5
	Blank	0	0
H3B1B2		129	129
	Absentee	17	17
	Provisional	5	5

VOTE TOTALS

President	
PresidentC1 (Moderate)	55
PresidentC2 (Progressive)	46
PresidentC3 (Action)	27

Overvotes	0
Undervotes	1
Counted ballots	129
Balance	0
U.S. Senate	
USSenateC1 (Moderate)	66
USSenateC2 (Progressive)	62
Overvotes	0
Undervotes	1
Counted ballots	129
Balance	0
U.S. House	
USHouseC3 (Conservation)	44
USHouseC1 (Moderate)	40
USHouseC2 (Progressive)	40
Overvotes	0
Undervotes	5
Counted ballots	129
Balance	0
State Senate	
SSenateC1 (Moderate)	43
SSenateC2 (Progressive)	41
SSenateC3 (Conservation)	41
Overvotes	0
Undervotes	4
Counted ballots	129
Balance	0
State House #3	
SHouseD3C1 (Independent)	42
SHouseD3 Writein1 (write-in)	40
SHouseD3 Writein2 (write-in)	6
SHouseD3 Writein3 (write-in)	4
Invalid Writein (write-in)	4
Overvotes	0
Undervotes	33
Counted ballots	129
Balance	0
County Council	
CCouncilC03	44
CCouncilC05 (Progressive)	32
CCouncilC07 (Moderate)	28
CCouncilC10 (Progressive)	28

CCouncilC01 (Moderate)	19
CCouncilC02 (Progressive)	17
CCouncil Writein1 (write-in)	17
CCouncilC08 (Independent)	14
CCouncilC06	10
CCouncilC09 (Conservation)	10
CCouncil Writein2 (write-in)	10
CCouncilC04 (Conservation)	8
CCouncil Writein3 (write-in)	7
CCouncil Writein4 (write-in)	4
CCouncil Writein5 (write-in)	4
Invalid Writein (write-in)	3
Overvotes	116
Undervotes	145
Counted ballots	129
Balance	0
Ballot Initiative 1	
Yes	64
No	49
Overvotes	0
Undervotes	16
Counted ballots	129
Balance	0
Ballot Initiative 2	
Yes	55
No	49
Overvotes	0
Undervotes	25
Counted ballots	129
Balance	0

6.6.4 Statehouse District results

Report for context StateHouseD1 generated 2008-03-11 15:48:09-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		566	562
	Absentee	79	78
	Provisional	36	36
	Blank	7	7
H1B1B2		479	475
	Absentee	68	67

	Provisional	30	30
	Blank	5	5
H1B1B2M1B4		87	87
	Absentee	11	11
	Provisional	6	6
	Blank	2	2

VOTE TOTALS

President	
PresidentC1 (Moderate)	224
PresidentC2 (Progressive)	212
PresidentC3 (Action)	112
Overvotes	0
Undervotes	14
Counted ballots	562
Balance	0
U.S. Senate	
USSenateC1 (Moderate)	279
USSenateC2 (Progressive)	271
Overvotes	0
Undervotes	12
Counted ballots	562
Balance	0
U.S. House	
USHouseC2 (Progressive)	202
USHouseC3 (Conservation)	175
USHouseC1 (Moderate)	171
Overvotes	0
Undervotes	14
Counted ballots	562
Balance	0
State Senate	
SSenateC3 (Conservation)	185
SSenateC2 (Progressive)	177
SSenateC1 (Moderate)	175
Overvotes	0
Undervotes	25
Counted ballots	562
Balance	0
State House #1	
SHouseD1C1 (Moderate)	200
SHouseD1C2 (Progressive)	193
SHouseD1C3 (Conservation)	108

SHouseD1 Writein1 (write-in)	12
SHouseD1 Writein2 (write-in)	11
Invalid Writein (write-in)	7
Overvotes	0
Undervotes	31
Counted ballots	562
Balance	0
County Council	
CCouncilC03	191
CCouncilC07 (Moderate)	130
CCouncilC05 (Progressive)	111
CCouncilC10 (Progressive)	102
CCouncilC01 (Moderate)	88
CCouncil Writein1 (write-in)	85
CCouncilC08 (Independent)	75
CCouncilC06	74
CCouncilC09 (Conservation)	72
CCouncilC02 (Progressive)	71
CCouncil Writein2 (write-in)	47
CCouncilC04 (Conservation)	45
CCouncil Writein3 (write-in)	31
CCouncil Writein4 (write-in)	27
CCouncil Writein5 (write-in)	23
Invalid Writein (write-in)	6
Overvotes	372
Undervotes	698
Counted ballots	562
Balance	0
Ballot Initiative 1	
Yes	260
No	193
Overvotes	0
Undervotes	109
Counted ballots	562
Balance	0
Ballot Initiative 2	
Yes	247
No	225
Overvotes	0
Undervotes	90
Counted ballots	562
Balance	0
Ballot Initiative 4	
No	34

Yes	33
Overvotes	0
Undervotes	20
Counted ballots	87
Balance	0

Muni City Council	
MCCouncilC2	47
MCCouncilC5	30
MCCouncilC4	28
MCCouncilC3	10
MCCouncilC1	5
Overvotes	0
Undervotes	54
Counted ballots	87
Balance	0

Report for context StateHouseD2 generated 2008-03-11 15:48:09-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		695	686
	Absentee	82	81
	Provisional	40	39
	Blank	7	7
H2B1B2B3		547	540
	Absentee	61	61
	Provisional	33	32
	Blank	6	6
H2B1B2B3M1B4		148	146
	Absentee	21	20
	Provisional	7	7
	Blank	1	1

VOTE TOTALS

President	
PresidentC2 (Progressive)	290
PresidentC1 (Moderate)	267
PresidentC3 (Action)	118
Overvotes	0
Undervotes	11
Counted ballots	686
Balance	0

U.S. Senate	
USSenateC1 (Moderate)	334
USSenateC2 (Progressive)	331
Overvotes	0
Undervotes	21
Counted ballots	686
Balance	0
U.S. House	
USHouseC2 (Progressive)	246
USHouseC1 (Moderate)	211
USHouseC3 (Conservation)	208
Overvotes	0
Undervotes	21
Counted ballots	686
Balance	0
State Senate	
SSenateC2 (Progressive)	230
SSenateC1 (Moderate)	220
SSenateC3 (Conservation)	207
Overvotes	0
Undervotes	29
Counted ballots	686
Balance	0
State House #2	
SHouseD2C1 (Moderate)	269
SHouseD2C2 (Progressive)	266
SHouseD2C3 (Independent)	72
SHouseD2 Writein1 (write-in)	44
Invalid Writein (write-in)	2
Overvotes	0
Undervotes	33
Counted ballots	686
Balance	0
County Council	
CCouncilC07 (Moderate)	199
CCouncilC03	197
CCouncilC05 (Progressive)	126
CCouncil Writein1 (write-in)	114
CCouncilC01 (Moderate)	110
CCouncilC10 (Progressive)	106
CCouncilC02 (Progressive)	93
CCouncilC08 (Independent)	85
CCouncilC09 (Conservation)	81

CCouncilC06	73
CCouncilC04 (Conservation)	66
CCouncil Writein2 (write-in)	64
CCouncil Writein4 (write-in)	39
CCouncil Writein3 (write-in)	34
CCouncil Writein5 (write-in)	20
Invalid Writein (write-in)	11
Overvotes	560
Undervotes	766
Counted ballots	686
Balance	0
Ballot Initiative 1	
Yes	353
No	217
Overvotes	0
Undervotes	116
Counted ballots	686
Balance	0
Ballot Initiative 2	
No	303
Yes	280
Overvotes	0
Undervotes	103
Counted ballots	686
Balance	0
Ballot Initiative 3	
No	311
Yes	207
Overvotes	0
Undervotes	168
Counted ballots	686
Balance	0
Ballot Initiative 4	
Yes	60
No	52
Overvotes	0
Undervotes	34
Counted ballots	146
Balance	0
Muni City Council	
MCCouncilC2	68
MCCouncilC5	61
MCCouncilC4	37

MCCouncilC3	24
MCCouncilC1	8
Overvotes	0
Undervotes	94
Counted ballots	146
Balance	0

Report for context StateHouseD3 generated 2008-03-11 15:48:09-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		739	731
	Absentee	93	92
	Provisional	44	44
	Blank	5	5
H3B1B2		739	731
	Absentee	93	92
	Provisional	44	44
	Blank	5	5

VOTE TOTALS

President	
PresidentC1 (Moderate)	305
PresidentC2 (Progressive)	276
PresidentC3 (Action)	140
Overvotes	0
Undervotes	10
Counted ballots	731
Balance	0

U.S. Senate	
USSenateC2 (Progressive)	366
USSenateC1 (Moderate)	351
Overvotes	0
Undervotes	14
Counted ballots	731
Balance	0

U.S. House	
USHouseC1 (Moderate)	243
USHouseC3 (Conservation)	233
USHouseC2 (Progressive)	230
Overvotes	0

Undervotes	25
Counted ballots	731
Balance	0
State Senate	
SSenateC2 (Progressive)	240
SSenateC1 (Moderate)	232
SSenateC3 (Conservation)	228
Overvotes	0
Undervotes	31
Counted ballots	731
Balance	0
State House #3	
SHouseD3 Writein1 (write-in)	266
SHouseD3C1 (Independent)	221
SHouseD3 Writein2 (write-in)	41
SHouseD3 Writein3 (write-in)	20
Invalid Writein (write-in)	14
Overvotes	0
Undervotes	169
Counted ballots	731
Balance	0
County Council	
CCouncilC03	207
CCouncilC07 (Moderate)	182
CCouncilC05 (Progressive)	141
CCouncil Writein1 (write-in)	131
CCouncilC01 (Moderate)	124
CCouncilC10 (Progressive)	121
CCouncilC06	101
CCouncilC02 (Progressive)	77
CCouncilC09 (Conservation)	77
CCouncilC08 (Independent)	73
CCouncilC04 (Conservation)	65
CCouncil Writein2 (write-in)	54
CCouncil Writein3 (write-in)	39
CCouncil Writein4 (write-in)	36
CCouncil Writein5 (write-in)	19
Invalid Writein (write-in)	13
Overvotes	652
Undervotes	812
Counted ballots	731
Balance	0
Ballot Initiative 1	
Yes	374

No	253
Overvotes	0
Undervotes	104
Counted ballots	731
Balance	0

Ballot Initiative 2	
No	315
Yes	305
Overvotes	0
Undervotes	111
Counted ballots	731

6.6.5 Municipality results

Report for context Municipality1 generated 2008-03-11 15:48:07-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		235	233
	Absentee	32	31
	Provisional	13	13
	Blank	3	3
H1B1B2M1B4		87	87
	Absentee	11	11
	Provisional	6	6
	Blank	2	2
H2B1B2B3M1B4		148	146
	Absentee	21	20
	Provisional	7	7
	Blank	1	1

VOTE TOTALS

President	
PresidentC2 (Progressive)	92
PresidentC1 (Moderate)	82
PresidentC3 (Action)	51
Overvotes	0
Undervotes	8
Counted ballots	233

Balance	0
U.S. Senate	
USSenateC2 (Progressive)	113
USSenateC1 (Moderate)	110
Overvotes	0
Undervotes	10
Counted ballots	233
Balance	0
U.S. House	
USHouseC2 (Progressive)	90
USHouseC3 (Conservation)	67
USHouseC1 (Moderate)	66
Overvotes	0
Undervotes	10
Counted ballots	233
Balance	0
State Senate	
SSenateC3 (Conservation)	79
SSenateC1 (Moderate)	73
SSenateC2 (Progressive)	72
Overvotes	0
Undervotes	9
Counted ballots	233
Balance	0
State House #1	
SHouseD1C1 (Moderate)	31
SHouseD1C2 (Progressive)	26
SHouseD1C3 (Conservation)	20
SHouseD1 Writein1 (write-in)	3
SHouseD1 Writein2 (write-in)	3
Invalid Writein (write-in)	0
Overvotes	0
Undervotes	4
Counted ballots	87
Balance	0
State House #2	
SHouseD2C1 (Moderate)	61
SHouseD2C2 (Progressive)	58
SHouseD2C3 (Independent)	11
SHouseD2 Writein1 (write-in)	10
Invalid Writein (write-in)	1
Overvotes	0
Undervotes	5

Counted ballots	146
Balance	0
County Council	
CCouncilC07 (Moderate)	67
CCouncilC03	64
CCouncilC05 (Progressive)	51
CCouncilC09 (Conservation)	37
CCouncilC10 (Progressive)	37
CCouncil Writein1 (write-in)	37
CCouncilC01 (Moderate)	33
CCouncilC08 (Independent)	29
CCouncilC02 (Progressive)	26
CCouncilC04 (Conservation)	23
CCouncilC06	22
CCouncil Writein2 (write-in)	20
CCouncil Writein4 (write-in)	12
CCouncil Writein3 (write-in)	8
CCouncil Writein5 (write-in)	6
Invalid Writein (write-in)	5
Overvotes	208
Undervotes	247
Counted ballots	233
Balance	0
Ballot Initiative 1	
Yes	119
No	75
Overvotes	0
Undervotes	39
Counted ballots	233
Balance	0
Ballot Initiative 2	
Yes	103
No	96
Overvotes	0
Undervotes	34
Counted ballots	233
Balance	0
Ballot Initiative 3	
No	69
Yes	48
Overvotes	0
Undervotes	29
Counted ballots	146
Balance	0

Ballot Initiative 4	
Yes	93
No	86
Overvotes	0
Undervotes	54
Counted ballots	233
Balance	0
Muni City Council	
MCCouncilC2	115
MCCouncilC5	91
MCCouncilC4	65
MCCouncilC3	34
MCCouncilC1	13
Overvotes	0
Undervotes	148
Counted ballots	233
Balance	0

6.7 Conclusion

Any person interested in simulating potential results of a specific election and testing voting systems for use in that election may follow the procedure described in this section to generate sample ballots. This procedure helps to identify the features of a specific election that are relevant under the logic model of the VVSG and provides guidance in representing those features in the abstract and physical models of Votetest. The database representation can then be augmented with probability distributions that are most relevant to the local election and sample ballots generated.

The sample ballots can be input to the voting system in use for the local election to verify that those systems operate as expected for that election. In addition, Votetest reporting results for the same sample ballots can be used to verify that the voting system being tested produces correct results for that election as required by the VVSG.

References

[1] Election Assistance Commission. *Proposed Draft Revisions to 2005 Voluntary Voting System Guidelines (VVSG v.1.1)*, May 27, 2009. http://www.eac.gov/program-areas/voting-systems/voting-system-certification/2005-vvsg/draft-revisions-to-the-2005-voluntary-voting-system-guidelines-vvsg-v-1-1.

[2] Information technology—Database languages—SQL. ISO/IEC 9075, International Organization for Standardization, 2003. http://www.iso.org/.

[3] Election Assistance Commission. *Voluntary Voting System Guidelines Recommendations to the Election Assistance Commission*, August 31, 2007. http://vote.nist.gov/vvsg-report.htm.

[4] Election Assistance Commission. *2005 Voluntary Voting System Guidelines*, Version 1.0, March 6, 2006. http://www.eac.gov/voting%20systems/voting-system-certification/2005-vvsg.

[5] OMG Unified Modeling Language specification, version 1.5. Document formal/2003-03-01, Object Management Group, March 2003. http://www.omg.org/cgi-bin/doc?formal/2003-03-01.

[6] Sample ballot collection. http://vote.nist.gov/ballots.htm.

[7] PostgreSQL version 8.3.7, March 17, 2009. http://www.postgresql.org/.

[8] Class Library for Numbers version 1.2.2, April 2008. http://www.ginac.de/CLN/.

[9] GNU Compiler Collection version 4.3.3, January 24, 2009. http://gcc.gnu.org/.

[10] Flex version 2.5.35, February 26, 2008. http://flex.sourceforge.net/.

[11] Bison version 2.3, June 2006. http://www.gnu.org/software/bison/.

[12] Programming languages—C++. ISO/IEC 14882, International Organization for Standardization, 2003. http://www.iso.org/.

[13] GNU Automake version 1.10.1, January 2008. http://www.gnu.org/software/automake/automake.html.

[14] Cygwin, December 2007. http://www.cygwin.com/.

[15] Installing PostgreSQL on Windows using Cygwin FAQ, October 2007. http://www.postgresql.org/docs/faqs.FAQ_CYGWIN.html.