Using TEVS for Vote Counting

Trachtenberg Election Verification System

by

Mitch Trachtenberg

1. INTENDED AUDIENCE

This document is aimed at users who are comfortable working on a command line or entering DOS commands; users should be comfortable using text editors to modify configuration files and should understand what a “directory tree” means. Ideally, the user is comfortable with UNIX and/or Linux.

If that’s not you, you should probably find someone to help you run this software.

The software enables anyone to conduct their own independent scan and vote count in a jurisdiction that allows this. If there are substantial differences between your vote count and the reported count, you can then investigate further.

It is a tool that will work well for a user who understands its operation and limitation, but it will just lead to confusion if used by someone who doesn’t understand these things. This is not “pick it up as you go along” software.

This is not security-oriented software. It is intended to be received from a trusted source and installed on a machine that is not networked, is physically secure, and used by trusted personnel. You have access to the complete “source code,” so you must decide for yourself whether it is doing what it is intended to do.

If you have downloaded this software as an ISO file to be burned to disk, you MUST do the following check to ensure that the file has arrived unaltered. From a Linux machine (or another machine capable of running the openssl command), type the following into a command window:

```
openssl dgst -ripemd160 tevs.iso
```

Then, contact your trusted provider and read them the output of this command. It should match their known output.

This is the ONLY way to ensure that you are receiving what your trusted provider has made available.

DO NOT RUN THE OPENSSL COMMAND ON A DISK YOU HAVE GENERATED FROM THE ISO AS THIS WILL DEMONSTRATE NOTHING ABOUT THE AUTHENTICITY OF THE DISK. DO NOT RELY ON THE WEB SITE OF THE TRUSTED PROVIDER -- TELEPHONE A PERSON KNOWN TO YOU.

SECURITY IS ONLY AS STRONG AS THE WEAKEST LINK. IF YOU USE AN UNVERIFIED COPY OF THIS SOFTWARE, ALL RESULTS MAY BE FRAUDULENT. DO THE CORRECT CHECK AT THE BEGINNING OF YOUR PROCESS, OR THE REST IS NOT WORTH BOTHERING WITH.

The security benefit of this software resides in its offering an independent second-opinion. It is my belief that it is much harder to defraud a count, or submit a suspicion-free but erroneous count, when a count is performed by two or more systems, at least one of which is “open source.”

This software does not know, and therefore cannot tell you, that it has been given all the ballots that voters cast, as they cast them. All it can do is scan the ballots it is given and count votes from the resulting images.

2. OVERVIEW

OK. Not scared away yet?

2.1 TEVS counts votes from ballots

TEVS (Trachtenberg Election Verification System) is a tool for those who want to count votes from ballots. Instead of trying to handle every vote on every ballot, it just wants to get most votes from most ballots, and lets its owner know what work remains to be done.

Simply running TEVS will get you an independent count, and allow you to either accept the official count or, if the percentages are substantially different, allow you to investigate further.

2.2 TEVS is Transparent

TEVS makes it possible for an audience to watch the ballot scans as they are created, seeing the scans overlaid with TEVS interpretation of what candidates have been voted for. It generates intermediate data that can be imported into spreadsheets or databases, so that final counting can be done by other systems. It can generate extensive logs showing what it is doing. It is designed to be the opposite of a "black box."

2.3 TEVS is Free and Open Source

TEVS is distributed installed on Ubuntu Linux. Both TEVS and Linux are free software meaning anyone may copy and distribute it to anyone else. Free means "free as in freedom." Anyone may charge for TEVS, but they cannot prevent you from getting the system from someone else. No one can legally improve the system and then "lock up" the improved system by selling it as non-free software. For more information, see the GNU General Public License version 3. Besides a bit of ballot-specific image processing, TEVS is mostly just "glue" that calls upon other free software to do what needs to be done.

Both TEVS and Linux are "open source," meaning you can examine, line by line, the program that is being run by your computer. It also means you (or anyone) can generate modified versions. This ability for anyone to modify the software allows for rapid evolution. It means that a worldwide community of interested programmers can potentially respond to bugs and insert improvements. It also means you must either trust that you have an unaltered version or must determine for yourself that the version you have is not doing anything out of the ordinary. If you understand digital signatures, you can confirm for yourself that the version you use comes from a source you trust.

The TEVS installation, complete with Ubuntu Linux, can be used to replace your regular hard disk, so you don't need to worry about interaction between TEVS and your computer's regular environment.

TEVS is written in the computer languages Python and C.

2.4 TEVS is supported

I provide fee-based support for this software, and will fix and extend problems for paying customers. Others are welcome to do the same; because the software is free and open source, programmers should be able to modify TEVS without any need to contact me, if they so choose. I can be contacted at mjtrac@gmail.com or via http://www.mitchtrachtenberg.com.

2.5 What TEVS does

Given a ballot of a general design TEVS knows about (for example, a Hart, Premier/ Diebold, or ES&S ballot meeting common criteria), TEVS analyzes the ballot to locate the contests, candidates, and “vote opportunities” -- the boxes or ovals or lines that a voter can mark to indicate a vote. When it encounters a new ballot layout (a new precinct, for example, or a new party ballot), it analyzes the layout, reads the text, and saves the layout and text to use with subsequent ballots of the same precinct and party. Ballots do not need to be batched by precinct or party; TEVS re-reads the correct layout every time it encounters an instance of a layout it has already encountered.

(Note: TEVS can be modified to handle ballot types it doesn’t yet understand, but this will require additional programming.)

To enable high-quality optical character recognition, TEVS needs to start by scanning a blank ballot of each party/precinct combination at a high resolution. Scanning of remaining ballots can then take place at a lower resolution. Typically, the blanks need to be scanned at 300 dots per inch or better, and the remaining ballots can be scanned at 150 dots per inch, which is much faster.

If you have no access to blank ballots or to ballot print files, you can still use TEVS but will need to check TEVS’ generated XML layout files -- it may be necessary add some information.

TEVS includes three main subsystems which may be run sequentially, simultaneously, or independently. Because the subsystems’ only connection is the files they write and read, any subsystem may be replaced by any process that works from the same input and generates the same type of output files.

2.5.1. Scan The first subsystem controls a scanner which is compatible with the SANE protocol. All it does is scan ballots into numbered files organized in the directory structure expected by the other subsystems. If you already have access to appropriate image files, you can simply place them into the unproc directory tree.

2.5.2. Count The second subsystem examines the images, determines the particular layout and candidate text, and counts votes for candidates off the images. For each votable area, it writes what will be referred to as a cast vote record, or CVR. Each CVR is a line containing comma-separated values indicating the ballot image, the area examined, which contest and candidate the area represents, whether the software thinks the area contains a vote, and various statistics that indicate why the software thinks what it thinks.

These line can easily be imported into almost any spreadsheet or database program for further processing.

For the current version, TEVS simply relies on the average lightness or darkness of vote opportunities to decide which opportunities contain votes. Because TEVS is open-source, it is easy for an experienced programmer to add more sophisticated analysis for borderline cases.

TEVS can generate extensive log files showing how it has generated the CVRs.

2.5.3. Show

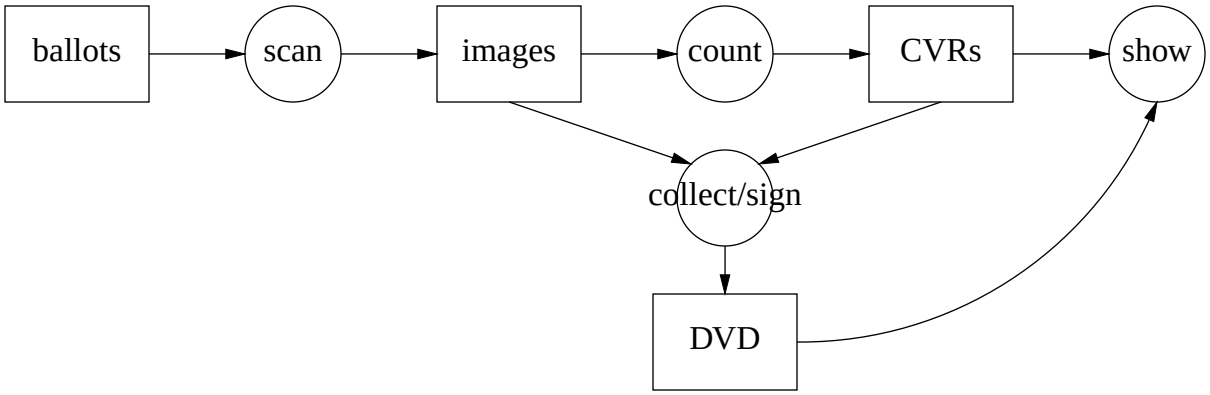
The third subsystem is a display program that shows each ballot together with an overlay showing the cast vote records as boxes around votable areas and text indicating the candidate or option represented by the boxed votable area. The text is colored to indicate whether the votable area was believed to contain a vote.

In addition, TEVS performs other associated tasks.

2.5.4. Summarize In addition to these three major components, the system is able to take all the CVRs and generate totals for candidates and options... the stuff you expect will get reported on election night. But you can also take the CVRs, import them into a spreadsheet or database, and do your own summaries, statistics, etc...

2.5.5. Archive You can and should ensure that the scanned images are saved in a secure manner. The best way to accomplish this is to use an encryption program to “digitally sign” the collection of images. Then, as long as your digital signature can be trusted, those who check can confirm that their copy of your image collection matches the original images. Ubuntu Linux contains the needed encryption program, gpg, as well as an easy-to-use front-end named “Seahorse.”

To confirm that the images match the original paper ballots, you should write your image collection to write-only storage like a DVD, and then compare a random sample of the written images from the DVD against the original paper ballots. (Even if you are using what you feel is a fresh copy of the correct software on a store-bought machine, scanning images on a store-bought scanner, having an outside observer perform this sampling before ballots are locked away will allow others to have greater confidence in the accuracy of the images.)



TEVS can generate collections of special-case ballot areas. It can write images of each marked write-in, to simplify counting of write-in candidates. It can write images of borderline votes, to simplify the task of obtaining final, exact numbers. And it will flag situations in which too many votes seem to have been entered in a contest, so that humans can decide which choice, if any, has really been voted for.

3. WHERE THINGS GO

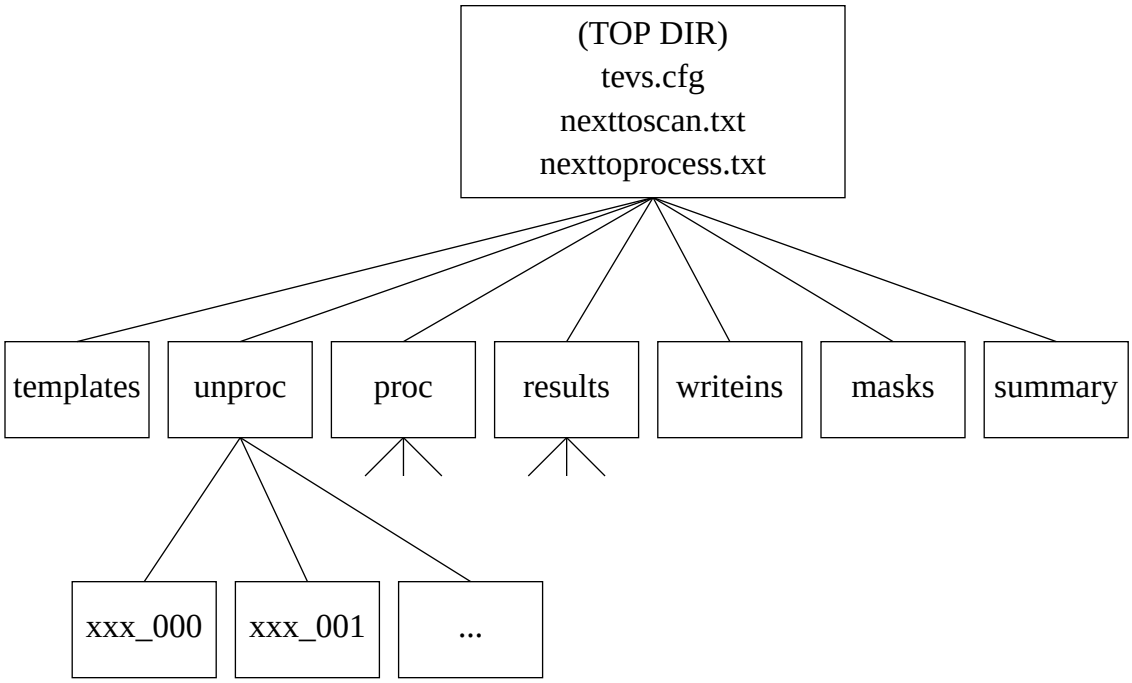


Image files are scanned (or placed) into subdirectories of unproc.

Once the files have been used to generate CVRs, they are moved to proc and CVRs are

placed in results.

Each of proc, unproc, and results contain subdirectories containing, in general, 1,000 files. Image files are named with a three letter prefix, an underscore, a zero-padded six digit serial number (starting with 1), and an extension, typically ".jpg". The first 999 images are written to ./unproc/xxx_000, the next 1,000 are written to from machine to machine, so that it is easy to merge the various directories generated by different machines operating in parallel.)

Image serial numbers begin with 1; ballots printed on two sides should generate an odd serial number for the front image and an even serial number for the back image.

Marked write-in vote opportunities are saved in ./writeins. Each writein file image is named with a combination of the name of the contest and the name of the image file from which it was cropped. which it was cropped

Templates for a particular party/precinct combination are stored in ./templates. Template names are based upon the bar code, dash pattern, or box pattern that was used to determine the template represented a new layout, precinct, or party. Template names have no extension, and contain XML.

The system may generate special versions of the template images that can be used to locate ballots with marks falling outside the vote opportunities (for example, a user writing "YES" next to a vote opportunity). These are stored in ./masks.

Summary results are written to ./summary.

The configuration file, allowing some modification of these various locations and other main characteristics of the system, is tevs.cfg. The number of the next image file that will be generated by the scanning is on a single line in nexttoscan.txt. The number of the next image file to process is on a single line in appropriate programs, and a GUI to adjust them when necessary will probably be added to the system; for now, any text editor can be used.

The main Python program code is located in . -- this will probably change. The Python code relies on a modified version of the Python Image Library, which is located in ˜/Imaging-1.1.7

4. RUNNING SUBSYSTEMS FROM THE COMMAND LINE

Although a central GUI will be provided to enable push button operation, TEVS subsystems are just scripts that can be run from the command line. Scanning is controlled from

./scanloop.py; counting (or data extraction) from as soon as the scanning program has placed some files in ./unproc; by default, the extraction program will exit as soon as the next sequential file is not found. The display program can be run at any time, but it can only display files that have been processed and moved to the ./proc tree, and for which a results file has been written to the ./results directory.

The display program cannot alter the data in the ./results directory, and so nothing you do with the display program will affect the data that has been extracted from the images.

Once the extraction program has processed all image files, ./summarize.py can be run to merge all results and generate totals for the contests and candidates.

4.1 Scanning

“python scanloop.py” runs a GUI which invokes the non-GUI python script scanctrl.py. scanctrl does the actual scanning using the SANE libraries. Each time scanctrl ends, scanloop can restart it, so that if you have a scanner which does not need to raise and lower an elevator, you can simply drop a new ballot into the scanner and it will scan it immediately.

scanloop can scan templates or ballots. The only difference is that templates are scanned at 300dpi (or greater), while ballots are scanned at the lower resolution of 150dpi. Templates should be blank ballots -- voted ballots may not generate proper templates, as vote opportunities may be missed. The best source of template images is not your scanner, but the print files which were used to print ballots. If you can obtain these, they can be used directly by splitting the pdfs into individual pages, converting the pages to 300 dpi jpgs, and numbering these jpegs as though they had been scanned.

4.2 Counting

There is no GUI associated with extracting the votes from the images: you simply run the command “python extraction.py”. As extraction proceeds, however, you can simultaneously run the command “python TEVS.py” to display the ballots which have been scanned and the votes which have been read.

4.3 Showing

“python TEVS.py” runs the ballot browsing subsystem. The system will display the ballots

sequentially when the “auto-advance” toggle button is set. You can unset the “auto-advance” toggle and jump directly to any image number by entering it in the number entry field and pressing the Enter key or clicking the Go button. You can manually advance and back up by pressing the left and right arrow buttons. You can zoom in on the images by using the zoom slider. You can switch between images without name overlays and images with name overlays by selecting the Edit --> whatever option, or pressing Ctrl-whatever. Not Yet Implemented: Hovering the mouse over a particular vote opportunity will display all statistics about that vote op. Not Yet Implemented: Click in the annotate area to store ballot specific information without altering any results; the annotations will reappear when the ballot is viewed, and an annotation file can be output that will show all notes entered. By selecting “Only annotated ballots”, the next and previous ballot buttons will go only to annotated ballots. Not Yet Implemented: By selecting “only ballots with code:” and entering a code, the next and previous ballot buttons will go only to ballots using the entered code. Same for “only ballots of party”, “only ballots with light votes”, “only ballots with overvotes”, and “only ballots with writeins”

5. CONFIGURING TEVS

5.1 A sample configuration file

```
[Layout]

# select from Hart, ESS, Diebold

brand:Hart

[Scanner]

templatedpi:300

ballotdpi:150

duplex:True

[Sizes]

ballot_width_inches:8.5

ballot_height_inches:17.0
```

```
oval_width_inches:0.32

oval_height_inches:0.18

candidate_text_inches:0.43

vote_intensity_threshold: 128

[Paths]

# This section provides for some flexibility with regard to
the

# naming of the input and output directories and files.

# Note that it is important to have a different prefix for
every machine

# used in running a scan/count that is to be merged, so that
when directory

# trees are merged, each machine gets its own space in the
merged tree.

# For example, two machines using prefixes “abc” and “def”
respectively

# will generate directories like results/abc_000..abc_999 and

# results/def_000...def_999, which may be safely merged.

# prefix may not include the forward slash or any path separa-
tion character

prefix:tmp

# proc may not include the forward slash or any path separa-
tion character

proc:proc
```

```
# unproc may not include the forward slash or any path separa-
tion character

unproc:unproc

# results may not include the forward slash or any path sepa-
ration character

results:results

# masks may not include the forward slash or any path separa-
tion character

masks: masks

# in [proc|unproc|results|masks]formatstrings,

# “thousands” will be substituted with “%03d”, “units” with
“%06d”

# to generate a filename-generating string, which is then

# provided with the next file number to deal with in order to
generate

# filenames, like this: “formatstring” % (n/1000,n)

# The proc and unproc format strings should mirror one
another,

# as files are moved from unproc to proc.

procformatstring:/media/My    Book/hum20100608/%(proc)s/%(pre-
fix)s_thousands/%(prefix)s_units.jpg

unprocformatstring:/media/My                                  Book/
hum20100608/%(unproc)s/%(prefix)s_thousands/%(pre-
fix)s_units.jpg

resultsformatstring:/media/My                                 Book/
```

```
hum20100608/%(results)s/%(prefix)s_thousands/%(pre-
fix)s_units.txt

masksformatstring:/media/My  Book/hum20100608/%(masks)s/%(pre-
fix)s_thousands/%(prefix)s_units.jpg

logfilename:tevs_log.txt
```

5.2 Settings

6. UNDERSTANDING TEVS' LOGGED OUTPUT

7. HANDLING AMBIGUOUS VALUES

7.1 Overvotes

7.1.1. Overvotes "Overvotes" occur when it appears a ballot has too many votes in a given contest. Voters do sometimes make this mistake, but quite often what the system sees as an overvote can be resolved for one of the options by manually inspecting the ballot in question.

To display the ballot in question, run the display program and type the image number (which is part of the image name, the first part of the CVR) followed by the enter key. Note the correct resolution. Depending on the procedure you wish to follow, you can directly change the overvote in the cast vote record, or simply maintain a list of resolutions.

7.1.2. Borderline Cases In an ideal world, vote opportunities with an average lightness below a particular value would always represent votes, and those with an average lightness at or above that value would always represent nonvotes.

In practice, this applies to the vast majority of vote opportunities.

Unfortunately, it does not apply to all. Some vote opportunities have what are called "hesitation marks," where a voter may have rested their pencil or pen. Others may contain surprisingly light checks, instead of being filled in.

As it counts, TEVS logs each ambiguous vote opportunity. TEVS can, if you wish, generate mapped mosaics showing all vote opportunities with ambiguous values. If all you need is to confirm the percentages of a contest, you do not need to generate or analyze these mosaics, but if you want to get a final count, each such vote opportunity must be classified as a vote or non-vote.

This process will gradually become more and more automated as TEVS evolves.

8. RUNNING TEVS VIA THE GRAPHICAL INTERFACE

9. GENERATING GRAPHICAL AND TABULAR REPORTS

10. MODIFYING TEVS

10.1 TEVS Files and Processing Flow

Files are currently located in /usr/local/lib/python2.6/dist-packages/tevs, which will be referred to as TEVS.

TEVS has python files in subdirectories ballottypes, utils, and main. In addition, TEVS has subdirectories sample and docs. Documentation for any extensions should be added to docs; samples exercising any extensions should be added to samples.

TEVS basic processing loop is as follows:

1. Read one or two images
2. if necessary, determine the ballot type being represented
3. create an instance of the appropriate subclass of Ballot
4. using the rules of the appropriate subclass, locate two or more landmarks on the images so that the images can be oriented
5. using the rules of the appropriate subclass, determine the “layout code” found on the images; this will enable you to look up the appropriate layout(s) if they are already cached, or build and cache new layouts if necessary
6. capture statistics necessary to determine votes, based on the images and the appropriate layouts
7. use the layouts to write each vote, capturing the current jurisdiction, contest, candidate or option, and statistics

The Ballot class lays out the sequence in which these functions are called, along with customization pre-functions and post-functions. Subclasses of Ballot may implement versions of the required functions. Therefore, adding a new ballot type generally consists of creating new versions of functions needed to accomplish the tasks outlined above.

Extensions can also be added to the reporting mechanism, so that it can report in forms other than per-ballot csv files.

“Ballot Class Functions”

GetLandmarks retrieves orienting information required by subsequent functions. The Ballot instance must have im1 and optionally im2 set to PIL images of ballot faces prior to calling GetLandmarks.

GetLandmarks will analyze im1 and im2 to determine the dpi and save it as self.dpi and self.dpi_y.

GetLandmarks will then locate two or more landmarks near the corners of the ballot images. It will store information about the actual x and y offsets of these landmarks in order to be able to translate the upper left landmark of any two ballots into alignment, and will generate a rotation needed to bring the tilt of this ballot instance in line with the tilt of any other.

Where GetLandmarks is unable to determine this information, it will write the ballot’s filename to a log file with an explanation as to why the ballot was not otherwise processed. It will then raise a BallotException, which will terminate the processing of the ballot.

GetLandmarks should also ensure that the image representing the front face of the ballot is im1 and that representing the back face of the ballot is im2, and should flip the images to accomodate upside down scans.

GetLayoutCode cannot be called until successful completion of GetLandmarks.

GetLayoutCode generates a number that uniquely identifies the particular ballot face that is in the front image of the ballot. The approach will vary widely from vendor to vendor. For Hart Ballots, the upper left barcode should be sufficient. For Diebold Ballots, the bottom dash strip should be sufficient.

Each implementation will depend upon the location and characteristics of the region to be analyzed.

Each implementation should set the ballot instance's layout_code attribute to an integer and should then give the value of self.layout_code as its return value. It may also set self.code_string to a text representation of the integer.

ValidateLayoutCode, if present, is a method which determines whether the layout code is acceptable and returns True if it is or False if it is not. When it deems a layout code invalid, it must write the code and filename to a log file and terminate further processing of the ballot. Typically, an invalid layout code will occur when the layout code area of the ballot image is damaged.

GetFront(Back)Layout searches the Ballot class' front(back)_dict dictionary using the layout_code (or code_string) as a key. If a layout is found, it is retrieved as self.front(back)_layout. If not, BuildFront(Back)Layout is called.

Layouts are generated in XML format, with a hierarchy of BallotSide, Contest, and Oval nodes. (Oval name must change to VoteOp or VOP.)

BallotSide attributes include dpi, precinct, lx (landmark x), ly (landmark y) and rot (rotation).

Contest attributes include prop (boolean indicating whether this contest has human candidates or is a proposition or measure), text indicating the title text for the contest, x, y, x2 and y2 indicating a bounding box for the contest on the image, and maxv, indicating the maximum number of votes allowable in the contest.

Oval attributes include text, indicating the choice text for the oval, x and y indicating the location of the upper left corner of the vote opportunity.

Once built, layouts are stored in the front(back)_layout_dict dictionaries, and are optionally preserved in long-term storage in order to allow them to be read back in at the start of any subsequent session. If layouts are preserved, it is important to provide the user an option for clearing all preserved layouts.

CaptureVoteInfo captures information about regions of the images of the current ballot using the stored template information corresponding to the current ballot's code. The information is stored in self.results as a list of VoteData instances, VoteData being a class acting mostly like a C structure.

If an implementation provides one or more entries in the Ballot class variable fine_adjust,

each entry is treated as a method which may be called on the class instance. The method is handed x and y coordinates believed to represent the current vote opportunity, and the method returns updated coordinates which may more accurately reflect the vote opportunity. for the

Sufficient information must be stored in each VoteData instance that a subsequent program can determine the image file from which the vote opportunity was analyzed, the text corresponding to the contest, the text corresponding to the choice, the coordinates of the choice, and the relative lightness or darkness of the area whose upper left corner is represented by the stored coordinates. Additional information may be stored by implementations by subclassing VoteData, which will need to provide functions able to interpret the additional information in their subclass.

WriteVoteInfo takes information from a list of VoteData, and converts it to a series of CR-delimited CSV records in a single character string, which is returned. No actual file is written by this function, but the returned string may be written to an external file to make it persistent.

ValidateLayoutCode to optionally test a given layout code to determine if it is acceptable for this election. A false return can trigger an additional call to GetLayoutCode, with added optional arguments indicating the number of prior calls to GetLayoutCode for this image, the prior layout codes returned, and the reasons for prior rejection(s). Different levels of implementation of GetLayoutCode may simply fail if ValidateLayoutCode sends back a rejection, or may call upon different strategies for locating an acceptable layout code.

GetFrontLayout and GetBackLayout to retrieve previously stored layout information corresponding to a layout code; the layouts must contain sufficient information to allow a vote gathering function to locate all vote opportunities on the layout and merge statistics from each image with jurisdiction/contest/candidate information from the layout.

BuildFrontLayout and BuildBackLayout to construct and store layout information for newly encountered layout codes;

FineAdjustVote for taking coordinates provided by the layout, as adjusted by translation and rotation of the individual image, and returning further adjusted versions of the coordinates that precisely line up based on local landmarks. This function is optional, called if registered, and can be chained in a list. The function returns None if there appears to be a problem in locating the vote target.

CaptureVoteInfo for using layout information to process the ballot image and extract vote

information;

WriteVoteInfo for writing the extracted vote information to a file and, optionally, for calling registered output functions to save vote information using alternative mechanisms.

11. THIS CAN'T WORK BECAUSE...

11.1 ...it will lead to vote buying and extortion.

Probably the most serious concern people express over making ballot images available online is that it enables people to place identifying marks or patterns on their ballot and have others view those marks or patterns.

This is true. As a simple example, fill out the upper left corner of two votes in an unimportant race to signal that this was their ballot. Most likely, no regular ballots will show that vote marking pattern.

This is an example of a more general problem that occurs any time even just the results of individual ballots are made public: if you are willing to fill a particular vote pattern into races that you don't care about, you can use that pattern as a signal that you cast a particular ballot.

The problem grows when images are made available, but it really exists whenever single-ballot vote records are made available.

By opening this signalling mechanism, you are opening the theoretical possibility that a crooked voter could sell their vote or that a crooked person could force someone to vote in a particular way.

However, if you are in a vote-by-mail jurisdiction, it is far easier to participate in extortion/vote-buying by utilizing mail-in ballots.

Let's take the example of an abusive husband who wants to control the way his spouse votes. With ballot images on-line, he could write a program to scan through all the images, instruct his wife to put a particular mark on her ballot, and take action if the software doesn't find that mark. But with vote-by-mail, he can simply demand that he be present when his wife fills out the ballot. Which do you think is more likely?

"...we aren't allowed to number ballots."

Well, the lawyers probably meant that you shouldn't number ballots until they've been cast and randomized. But if you are worried about the law, just don't number the ballots. Instead, set things up so that every tenth or twenty-fifth scan is a sheet with a number, and keep ballots in order. The files will be assigned numbers, and to track a file to a ballot you'll need to go to the closest numbered sheet, and then count forward or backward until you've reached the original scanned paper document.

"...we can't afford it." The cost of a scanner that can count 3,000 double sided ballots per hour is now under $8,000. The software is free. The system can run on a netbook which costs $300 and can be borrowed. You can assemble a bank of scanners able to scan at slower rates for about $500 per scanner. Please check with your elections office to ask how much it costs to print ballots for an election; you may be surprised.

"...the second count may be wrong, not the original count."

Yes, when you work with a redundant system, it is always possible that the backup system will be the one that fails. For truly mission-critical systems, engineers use three seperate systems and, when one disagrees with the other two, "majority wins" and the disagreeing system is taken off-line.

In vote counting, if the original system and the verification system disagree, you can go through all the intermediate results of the verification system. You'll either find where it went wrong, or you'll be able to satisfy yourself that its results are correct. With an official system that does not offer you intermediate results, you just have to take its word that its results are correct.

If a verification system is flat-out wrong more than once, you can bet it will stop being used, so the problem will solve itself. However, there will be borderline cases where "reasonable systems can disagree." It is useful for people to be educated about this, and being able to show typical instances of marginal votes -- something that you can do with TEVS -- can help with that voter education.

"...people will put false versions of the images on the net."

Digital signature technology is described by the American Bar Association as more secure

than regular signatures. By digitally signing a document, or a ballot collection, any person or group can provide any other person or group with a way of confirming that the document or ballot collection is still exactly the way it was when it was posted online.

If the concern is that online viewers will not be sophisticated enough to require validation of the digital signatures, one potential alternative would be making the ballot collections available only on DVDs mailed directly from the group or elections office.