Democracy Suite® ImageCast® C++ Coding Standard

Version: 5.11-CO::1

March 7, 2019

DOMINION VOTING

Our customers come first.

Table of Contents

Chapter 1: Introduction

1.1 Design Responsibility

Dominion Voting Systems (DVS) is the design authority.

1.2 Document Status

This is a working specification for discussion and analysis. Details are subject to change.

1.3 Patent Status

Certain system concepts, as well as many implementation and construction details, are protected by a series of U.S. and foreign patents pending.

1.4 Relevant Disclaimers

This document may make reference to certain Democracy Suite functionalities that are not part of the current 5.11-CO campaign and should be disregarded throughout the document.

For a full list of relevant disclaimers, please see the "Relevant Disclaimers" section in the *2.02 - Democracy Suite System Overview* document.

CHAPTER 2: C++ CODING STANDARDS

2.1 Obvious Constraints

- No self-modifying code
- No "exit" statements except from the main routine
- No "go-to" statements, or "while (0) {}" constructs
- Names used in code and in documentation must be consistent
- C++ language keywords shall not be used as names of functions, variables, structures, classes, etc.

2.2 Functions

- Functions must be clear and concise
- All functions must be preceded by a header. See section 2.11.4 Function Headers for details.

2.2.1 Function Names

- Abbreviations within function names are permitted, but their meaning should be clear

2.2.2 Input Parameters

- Parameter ranges must be validated on entry, or indicated in the header comments. If the caller guarantees that the parameters are within range, a comment stating so may instead be provided in the header.

2.2.3 Returns

- All non-void functions must have, at most, a single explicit "return" statement at the end. Midroutine returns should only be used for cases "...so severe that execution cannot be resumed".
- If a function detects an error and cannot complete, it may return a value indicating the error or throw an exception.
- Functions which return a pointer should return NULL (i.e. 0) to indicate failure.

2.2.4 Control Constructs

- Program flow should be based on simple combinations of the following constructs:
  - If-then-else
  - For-loop
  - Do-while
  - Do-until – Switch/case

2.3 Variables

Keep variables in the smallest possible scope, i.e. try to use locals where possible, and avoid using globals.

2.3.1 Variable Names

- Single character variable names should not be used, except for:
  - Loop variables, or
  - Mathematical or scientific standards, e.g. X and Y co-ordinates
- Variable names should include at least one noun or verb. Any tense or form is allowed, and abbreviations are permitted.
- The component parts of a variable name may be formed from whole words, abbreviations of words, or numbers, if they are significant to the meaning of the variable. Names may also contain the underscore character (“ ”), in order to separate the name into its component parts. See section 2.3.3 Compound Variable Names.
- Please take care to ensure that any variable name does not form a word that is offensive in any way.

2.3.2 Abbreviations

Variable names may be composed of full words, or abbreviations of full words. Two possible methods of abbreviating words are:

- Eliminate vowels, e.g. prnt can replace print
- Use the first three or more letters of the intended word, e.g. ball for Ballot

All abbreviations and acronyms used in variable names must be added to the DVS Glossary of Abbreviated Terms. This glossary should be maintained in the code repository of each project, and should be kept upto-date for reference by other programmers and code reviewers. A sample glossary is found in appendix A Sample Glossary of Terms.

2.3.3 Compound Variable Names

As mentioned above, variable names may be built-up of concatenations of several component parts. Each component part may be either a full word, or an abbreviation. At least one of the component parts should be either a noun or a verb (or an abbreviation of such). In order to make the variable name meaning more obvious, highlight where each component part begins. Use the following techniques:

- Capitalize the first letter of each component (after the first), or
- Separate component parts with an underscore character (“ ”)

Example: A variable containing the “Previous Rotation Association Found” could be either:

- prevRotAssocFnd, or
- prev rot assoc fnd

See section 2.3.4 Variable Declaration and Initialization concerning comments for variable names made of three or more abbreviations.

2.3.4 Variable Declaration and Initialization

Initialize every variable upon declaration, and comment its use. Variables which share the same meaning may share the same comment.

Member variables of a class must be initialized in the class constructor(s), either directly or indirectly, i.e. by calling a routine specifically intended to perform initialization. They must be commented in the header file that defines the class.

If a variable name contains three or more abbreviations, then the declaration comment must explain the full significance of each component abbreviation.

Example:

```
UINT32 prevRotAssocFnd=0; // Previous Rotation Association Found
```

2.4 Constants

Constants other than 0 and 1 should be defined or enum’d either in a header file, or the specific source file, if pertinent to that file only.

Use of non-defined constants, other than 0 or 1, must be clearly commented.

```
Names defined to replace constants should be intuitive.
```

2.5 Macros

- Macros cannot include a “return” statement or pass control beyond the next statement.

2.6 Conditionals

2.6.1 Explicit Comparisons

- Explicit comparisons are recommended, but not required. Both of the following are permitted:

```
if ( bValidResponse == TRUE ) // valid

if ( bValidResponse ) // valid
```

2.6.2 Embedded Assignment Statements

- Embedding assignments within a conditional is permitted, but only one per line

```
Example: The following requires two lines:

if ( ( (rc1 = myRoutine()) == SUCCESS ) && ( (rc2 = hisRoutine() )
== FAILURE ) )
```

2.7 Switch Statements

- All switch statements must explicitly include the “default” case.

2.8 Assert Statements

```
Use ASSERT instead of “assert”, which permits disabling of all
“asserts” in production code.
```

2.9 Error Checking

```
• The code must take specific precautions against unexpected
faults and memory corruption in the following areas specifically.
```

2.9.1 Bounds Checking

- Ensure array subscripts and pointer variables are within range, so that arrays, strings and dynamically allocated memory accesses are all valid. If all calling routines ensure incoming parameters are within range, it is not necessary for the called routine to do so, but then the function header comments must indicate this.
- Specifically, all calls to "malloc" must check for a `NULL` (0) returned value.
- Pointers may be initialized to `NULL` (0) or to the intended memory location.

2.9.2 Counter Overflow

- Specifically noted in the testing guidelines: Code must explicitly test to ensure that "vote counters" do not overflow. "Assuming the counter size is large enough… is not adequate".

2.10 Code Simplicity / Legibility

- The code must be easily understood by the personnel at the lab doing the certification. These points are meant to make their job of understanding our code easier.

2.10.1 Line Count

- A "line" here is defined as an executable or control line plus its formatting and comment lines.
  - No more than 50% of all functions should exceed 60 lines in length.
  - No more than 5% of all functions should exceed 120 lines in length.
  - No functions should exceed 240 lines in length. If necessary to exceed, justify in comments.

2.10.2 Indirection

- Do not use more than four levels of indirection Example:

```
a -> b -> c -> d [e] // too complex
```

2.10.3 Nesting

- Do not use more than five levels of indented scope.

2.11 Formatting

- Code must be formatted according to the following rules, and contains headers as described below:

2.11.1 Code Formatting

- All code lines must be less than or equal to 120 characters in width. Place comment lines ahead of statements if too wide.
- All module header line comments for Files, Classes and Function headers must be less than or equal to 120 characters in width.

2.11.2 File Headers

- State the purpose of the unit and how it works
- Include the date of creation and revision record
- Use the following sample as a template for file headers:

```
/*****************************************************************

**

** THE FILE CONTAINS PROPRIETARY INFORMATION

**

** The material and information contained herein may not be
reproduced, copied, ** printed or disclosed for any purpose to any
person or organization, except

** as may be consistent with the terms and conditions of a license
agreement or

** non-disclosure agreement by Dominion Voting. All copies must
contain this

** trade secret warning. The material contained herein is the
property of

** Dominion Voting and is considered confidential and proprietary
information.

**

** Dominion Voting Systems Corporation

** Toronto, Ontario, Canada

**

*****************************************************************
*************/ /
*****************************************************************
*************

**
```

```
**File name:vscan_drv.cpp

**Module name: SCANNER (Layer2)

**Description: - All scanner transport controls

**- Acquire images from CSIs (front & back)

**

**Author:Shandon Wong

**

**Revision History

**yyyy.mm.dd AuthorDescription

**---- -- -- -------- -------------------------------------------
------

**2006.12.14 JRBExtracted from CF200 code

**2007.01.03 JRBAdded return codes for all routines

**2006.01.14 Shandon ICP-1035 - incorrect return code from scan_it

**

*****************************************************************/
```

2.11.3 Doxygen Compliant File Headers

- Proprietary information (mandatory)
- \file - Name of the file (mandatory tag)
- \brief - Brief description (mandatory tag)
- Author: - Name of the person who performed last modification on file (mandatory)
- Revision: - SVN version of last modification on file (mandatory)
- Date: - Date and time of file last modification (mandatory)
- \verbatim - Log of file changes (revision history) (mandatory tag)

```
/
*****************************************************************
*********

**

** THE FILE CONTAINS PROPRIETARY INFORMATION

**
```

```
** The material and information contained herein may not be
reproduced, copied, ** printed or disclosed for any purpose to any
person or organization, except

** as may be consistent with the terms and conditions of a license
agreement or

** non-disclosure agreement by Dominion Voting. All copies must
contain this

** trade secret warning. The material contained herein is the
property of

** Dominion Voting and is considered confidential and proprietary
information.

**

** Dominion Voting Systems Corporation

** Toronto, Ontario, Canada

**

*****************************************************************
*************/ /*!

\fileInitialVerifier.h

\briefInitial verifier declaration

$Author: bill.smith $

$Revision: 9 $

$Date: 2015-11-06 13:06:29 -0500 (Fri, 06 Nov 2015) $

*

\verbatim

Revision History

yyyy.mm.dd AuthorRevision Description

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


 *2010.01.14 pirke    3692   Class InitialVerifier added
 *2010.01.14 pirke    3700   MachineContext initialization moved to
 *                           InitialVerifier class
 *2010.01.14 pirke    3729   ProgressList inherits from
                             StringAsArgument class.
```

```
2010.02.05 pirke     4180   Machine behavior settings persistence
\endverbatim
*/
```

2.11.4 Function Headers

Use the following as a template for all functions with greater than, or equal to, 10 lines of code.

NOTE: If no Globals are used in the module, do not include the line Globals Used. Similarly, if no Future Enhancements are envisioned, or no File Access is used, do not include the lines indicating their non-existence. All other fields are mandatory.

```
/*****************************************************************

**

** Function: ScanDivert

**

** Purpose:Send ballot through the Divertor Slot

**

** Description:

** Advance ballot until tail end is beyond Paper Sensor 4, (i.e.
the divertor slot). ** Then reverse, which should cause the ballot
to backup through the slot. While

** reversing, monitor Paper Sensor 3, if it becomes active, retry
the whole process.

**

** Future enhancement: Monitor Paper Sensor 5 to ensure ballot
drops

**

** Input:scan_data -> scan_data_t structure, for global scanning
parameters **

** Return: rc= VSCAN_OK = 0 => Success

**= VSCAN_DRV_PAPER_JAM_ERROR. This could be for 2 reasons: **1)
Advanced further than expected without PS4 becoming False **or 2)
Advanced OK, but reversing kept passing PS3.

**= VSCAN_DRV_ERROR => Low level Motor routine error.(rc from
ioctl)

**

** Globals used: gSecurity, gpSDManager
```

```
**

** File Access: The file specified to function scan_open is opened
and read.

**

** Call tree:

** ScanDivert

**| CDvsUARTDrv::CDvsUARTDrv

**| CDvsQSPI::CDvsQSPI

**| CDvsEdeviceDrv::CDvsEdeviceDrv

**| CDvsEdeviceDrv::Modem

**| CDvsEdeviceDrv::CreateBuffer

**| EDevice::SetDrv

**| EDevice::SetUartDrv

**| CDvsUARTDrv::~CDvsUARTDrv (Virtual)

**| CDvsQSPI::~CDvsQSPI (Virtual)

**| CDvsPrinterQSPI::~CDvsPrinterQSPI (Virtual)

**

** Revision History:

**yyyy.mm.dd AuthorDescription

**---- -- -- -------- -------------------------------------------
------

**2010.09.14 JRBInitial Revision

**

***************************************************************/
```

2.11.5 Functions Containing Less Than 10 Lines of Code

For functions less than 10 lines of code, either the above, or the following abbreviated header may be used.

```
/*****************************************************************
**
** Function: scan_divert
**
** Revision History:
**yyyy.mm.dd AuthorDescription
**---- -- -- -------- --------------------------------------------
------
**2010.09.14 JRBInitial Revision
**
*****************************************************************/
```

2.11.6 Doxygen Compliant Function Header

- Doxygen compliant function header:
- \fn - Function signature (mandatory tag)
- \brief - Brief description (mandatory tag)
- \details - Detail description (not mandatory tag)
- \param - Description of function parameter, this tag repeats for each function parameter. If function has no parameters this tag is omitted
- \return - Description of function return value (mandatory tag. If function has no return value add \return none.)
- \note Globals used: - List of used global variables in function. If no global variables are used this tag is omitted
- \note File access: - List of files accessed in function and method of access, i.e., read, write, modify or append. If no files is accessed this tag is omitted
- \verbatim - Function revision history and function call tree (mandatory tag)

Use the following as a template for all functions with greater than or equal 10 lines of code:

```
/*!
```

```
\fn bool FileSerializer::readDomDocumentFromFile(QDomDocument&
document,

const QString& fullPath)

*

\brief Read DomDocument from file

*

\details Function opens the file at specified full path and set is
as content to given

dom document

*

\param[out] document Document

\param[in] fullPath File path

*

\return Success flag

*

\note Globals used: gFile

*

\note File access: The file specified to function
readDomDocumentFromFile is opened* and read. *

\verbatim

Revision History

yyyy.mm.dd AuthorRevision

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

2010.02.10 pirke4254

2010.04.29 drazha5682

2010.05.25 jovica6080

*

Call Sequence

- QFile::open

- logQStringMessageDebug
- QDomDocument::setContent

- arg

- QFile::close

\endverbatim

*/
```

2.11.7 Doxygen Compliant Function Header for Functions Containing Less Than 10 Lines Of Code

For functions less than 10 lines of code, either the above, or the following abbreviated header may be used:

- \fn - Function signature (mandatory tag)
- \verbatim: - Function revision history and function call tree (mandatory tag)

```
/*!

\fn bool FileSerializer::readDomDocumentFromFile(QDomDocument& document,

const QString& fullPath)

*

\verbatim

Revision History

yyyy.mm.dd AuthorRevision

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

2010.02.10 pirke4254

2010.04.29 drazha5682

2010.05.25 jovica6080

*

Call Sequence

- QFile::open

- logQStringMessageDebug

- QDomDocument::setContent

- arg
- QFile::close
\endverbatim
*/
```

2.11.8 Class Headers

Headers for classes should use the following format:

```
/
*****************************************************************
***************
**
** Class:CDvsSecureIo
**
** Base Class:none
**
** Description:This class handles I/O to any file which is
encrypted or signed
**It also handles raw files (ie. neither encrypted nor signed)
**
**It supports most standard DvsFile I/O functions, with notable **
exception that a file cannot be opened for Read & Write, nor **can
one Seek in a file opened for Writing.
**
** Revision History:
**yyyy.mm.dd AuthorDescription
**---- -- -- -------- -------------------------------------------
------
**2010.09.14 JRBCreated
**
*****************************************************************
**************/
```

2.11.9 Doxygen Compliant Class Header

Headers for classes should use the following format:

- \class - Class name (mandatory tag)
- \brief - Brief description (mandatory tag)
- \details - Detail description (not mandatory tag) • \verbatim - Class revision history (mandatory tag)

/*!

\class M1Packageable

*****

\brief Base class for M1Package and M1Class.

*****

\details

Abstracts the fact that each M1Packageable element can be* packed into M1Package.

*****

\verbatim

Revision History

yyyy.mm.dd AuthorRevision

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

2009.11.06 pirke1711

2010.04.25 pirke5501

\endverbatim

***/**

2.12 Assembler Code

- Any Assembler routines must:
  - Have a single entry point and return
  - Follow “C” like control paths, if-then-else, do-while, etc
  - Be commented to read like “C” code

2.13 Classes with Code Fully or Partially Generated Using DVS Applifter Plug-in

- File header is located in Preserve section
- Whole code that should be intact after the next auto generating step is placed inside the Preserve section. Everything else is re-generated according to the model context.
- Preserve section of the Class' header contains description of class functionality
- Constructor's Preserve section contains addition to the Constructor code
- Class' Preserve sections (Public, Protected and Private) contain additional Functions declarations from the Class that are documented
- Class' Preserve section (File Footer) contains additional Functions definitions from the Class that are documented.
- Data members that are auto generated are not commented in source code.

2.14 Checklist

Following is a checklist for C++ code so that it meets our Coding Guidelines. Rules listed under "New Code" are not absolute requirements for Election Assistance Commission (EAC) certification, and so we need not modify existing code to comply with them. However any newly written code should include them.

2.14.1 All Code

- File and Function headers
- 120 column width
- variableNames > 1 char long & differ by > 1 character
- Variable declarations: Assign initial values and comment variables use
- #define or enum all constants (other than 0 and 1)
- Conditionals: max of 1 embedded assignment per statement
- “default” for all “switch” statements
- Use ASSERT instead of assert
- Validate parameter ranges, esp. array subscripts and pointers (or comment in function header)
- Test for divide by zero
- Test return values for errors
- No more than four levels of indirection
- No more than five levels of indent

Appendix A: Glossary of Terms

A.1 Computer Industry Abbreviations

Abbreviation	Meaning
arg	Argument
err	Error
ptr	Pointer
dbg	Debug
pkt	Packet

Table A-1: Computer Industry Abbreviations

A.2 Cryptographic Abbreviations

Abbreviation	Meaning
csum	Checksum
sig	Signature
asymm	Asymmetric
pkey	Private Key
enc	Encrypted

Table A-2: Cryptographic Abbreviations

A.3 Common Language Abbreviations

Abbreviation	Meaning
id	Identifier
app	Application
chk	Check
orig	Original
reqd	Required

Table A-3: Common Language Abbreviations

A.4 Computer Industry Standard Acronyms

Acronym	Meaning
FTP	File Transfer Protocol (networking)
AES	Advanced Encryption Standard (cryptography)
PCL	Printer Control Language (Hewlett Packard)
RGB	Red/Green/Blue (color separation)
LED	Light Emitting Diode (computer hardware device)
CR	Carriage Return (ASCII 0x0D - telecommunications)
GPIO	General Purpose Input/Output (computer hardware device)
IOCTL	I/O ConTroL (kernel call) (Linux kernel terminology)
CR	Carriage Return (ASCII 0x0D - telecommunications)

Table A-4: Computer Industry Standard Acronyms

A.5 Voting Industry Standard Acronyms

Acronym	Meaning
AVS	Accessible Voting Session
BMD	Ballot Marking Device
RCV	Rank Choice Voting

Table A-5: Voting Industry Standard Acronyms

A.6 Dominion Specific Acronyms and Terms

Acronym	Meaning
DVS	Dominion Voting Systems
VIF	Voting Information File
wmark	Watermark (applied over ballot image)
CF200	Tabulator product model

Table A-6: Dominion Specific Acronyms and Terms

A.7 Hungarian Notation Prefixes Indicating Variable Type

Prefix	Variable Type
m	Member
p	Pointer
b	Boolean

Table A-7: Hungarian Notation Prefixes Indicating Variable Type

REVISION HISTORY

Rev.	Date	Author	Summary
1	03-06-2019	brian.fitzsimmons	Created 5.11-CO branch from trunk

List of Figures

List of Tables

End of Document

DOMINION VOTING®

Our customers come first.