LimnoTech
Water Environment | Scientists Engineers

501 Avis Drive
Ann Arbor, MI 48108
734.332.1200
www.limno.com

Memorandum

From: Tad Slawecki

To: Junaid As-Salek (USBR)
Mike FitzMaurice (USBR)

Date: December 17, 2012
Project: CALGUI2
CC: Click here to enter text.

SUBJECT: **CalLite GUI Code Review**

Introduction

The CalLite GUI (CALGUI) is intended to improve accessibility and usability of the CalLite screening model for planning and management of the State Water Project and Central Valley Project in California. In work performed in 2010-2012, version 2.0 of CalLite and the accompanying CalLite GUI were developed, tested, and released by California Department of Water Resources and Bureau of Reclamation. An updated version of the CalLite model and GUI are currently under development; the initial task in this effort, reported on in this document, is a formal code review to assess and improve quality, clarity, maintainability and performance of the GUI.

CALGUI Overview

The current CALGUI implementation (revision 328, 2/10/2012) implements the conceptual design shown in Figure 1 with three key functionalities:

- On invocation, CALGUI builds an externally-reconfigurable GUI using GUI.xml and GUI_Links*.table files.
- The GUI is used by end users to create and run CalLite simulations (“scenarios”) that consist of an ensemble of .dss (binary) and .table and .sty (ASCII) files. In some files, particular values depend on user settings in the GUI, while the inclusion or exclusion of other .table files and of all .dss files are controlled by GUI settings. Scenarios are defined by (and saved as) the GUI state.
- The GUI is also used to view or compare simulation results from CalLite scenarios.

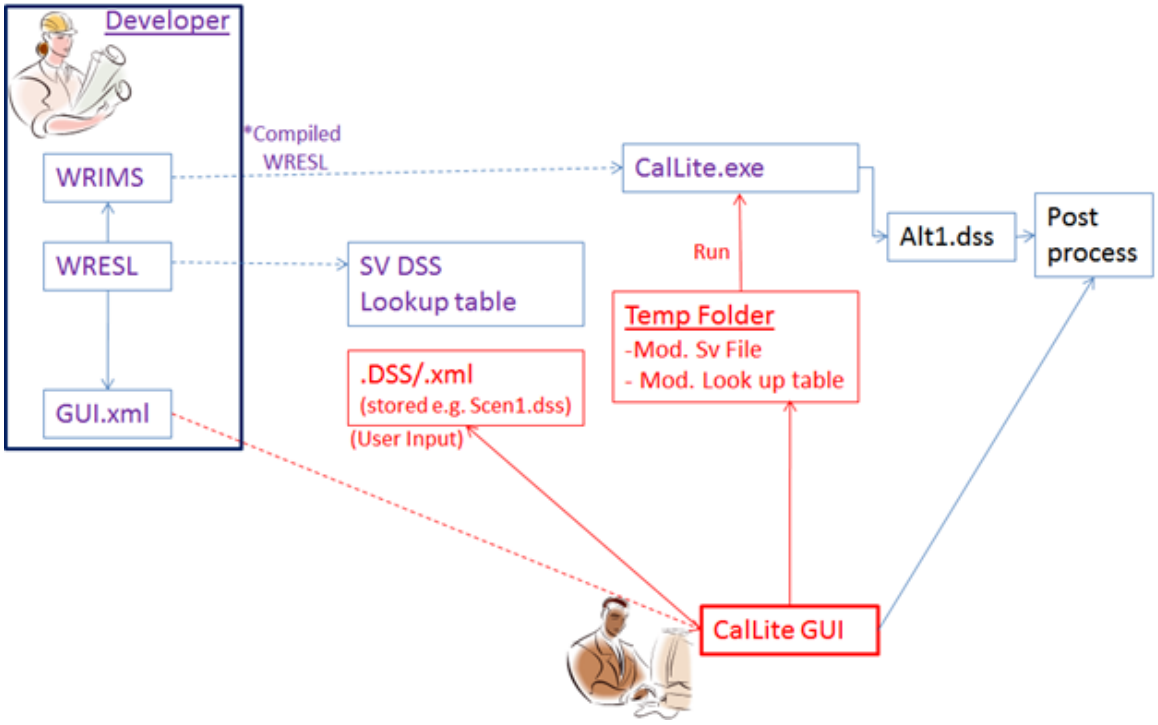

Figure 1. Initial conceptual design for CALGUI.

Review Process

The second phase of CalLite GUI development will implement additional user controls for scenario creation that correspond to added CalLite features, add new display capabilities, and address known issues in the initial release. The code review was undertaken to assess the status of the existing code base and identify actions to improve the quality and reliability of the code in advance of the code modifications necessary to implement new capabilities.

The code review, led by Soumya Sengupta and Idrissa Tiemogo (Michael Baker, Inc.) and Nicky Sandhu (California Division of Water Resources) touched on 20 individual topics (presented in Figure 2). Components, testing, and package structure received the most focus during the six-week review period, though all topics were considered.

Goal	Session(s)	Topic of Review
Review Code Quality	Architecture	• High level system architecture and the components involved • Are components loosely coupled? (E.g.: Use of interfaces.) • Is design flexible?
	Components	• **Are the components cohesive?** • **Algorithms, data structures used (E.g.: Use of collection libraries; use of language structures like generics, enhanced for loops etc.)** • **I/O techniques used (E.g.: Was NIO considered?)** • **Does the design account for performance?** • **Refactoring patterns that are generally used**
Review Code Management	Code organization	• Use of version control • **Package structure** • Configuration management
	Testing	• **Unit, integration and functional testing process**
	Build	• Build process • Use of continuous integration
	Deployment	• Deployment process
Review Code Clarity	Style standards	• Determine style standards followed
	Documentation	• *Architecture level* • *Code level* • *Process management level* • *Installation level*

Figure 2. Topics for code review.

Review notes and planned actions for each topic are presented in the following section ("Components of Review"), and recommended actions are compiled in the section ("Summary of Actions").

Components of Review

Review Code Quality

Architecture

- High level system architecture and the components involved

  Review: Documentation of code inadequate

  Actions: Draft high-level architecture document developed.

- Are components loosely coupled? (E.g.: Use of interfaces.)

  Review: Components are not loosely coupled – packages/classes are "overloaded"

  Actions: Some aspects of the strong coupling – particularly to the Swing components of the GUI – are an artifact of the original implementation which conflates view and model functionality in the SwiXml component, and will be resource-intensive to address directly. However, implementation of JUnit tests for key functionality will result in localized improvements towards loose coupling.

- Is design flexible?

  Review: The design is not clearly flexible, but is adequate to address expressed client needs as currently understood with regards to (1) adding simple controls using GUI.xml and GUI_Links2.table; (2) extending user input of tabular information; (3) supporting batch operation.

  Action: Key methods will be restructured to improve testability – particularly FileAction.setupAndRun and GUI/file handling of tabular data – and thereby will also improve design flexibility.

Components

- Are the components cohesive?

  Review: Components are often too complex and performing too many tasks.

  Action: As with the coupling of components, restructuring of key classes and methods to improve JUnit testability will improve cohesiveness of code.

- Algorithms, data structures used (E.g.: Use of collection libraries; use of language structures like generics, enhanced for loops etc.)

  Review: Initial review identified overuse of if-then-else as one symptom.

  Action: Some if-then-else constructs have been simplified. On an ongoing basis, the development team should continue to identify where collections and other innate Java constructs can be deployed for improvement to code. Examples include use of hash tables or enum for month lookups, switch as alternative to i-t-e.. In general, consider if standard Java structures, constructs, or algorithms could be used to simplify the code.

- I/O techniques used (E.g.: Was NIO considered?)

  Review: CALGUI I/O is either simple text-based I/O or binary (DSS) I/O performed under the control of a mature library.

- Does the design account for performance?

  Review: Performance of the CalLite GUI has not been indicated as a concern.

  Action: Continue to work on implementing effective code that is based on efficient use of standard Java constructs and libraries, which have been optimized for performance.

- Refactoring patterns that are generally used

  Review: Refactoring is highly desirable in many locations in the code due to tight coupling, inadequate cohesion, and algorithm selection/implementation. Also noted in review: many of the same local variables are declared in multiple methods within the same class. Exception handling in the code is not mature – generally limited to console warnings.

  Action: Identify appropriate places for and apply commonly used design patterns (model-view-controller), employ effective refactoring techniques (Fowler). Renaming, method extraction and some other simpler ones have been applied

successfully to parts of code; need to continue for remainder – looking for opportunities to further apply simple refactoring tools and to use more complex ones where appropriate. Use of class variables, utility classes, or getter/setters are recommended to address the multiple declaration of local variables.

Strategy for improving exception handling (when and what o log v. alerting user, etc.) should be adopted that is based on distinguishing what kind of follow-up is expected – some errors are simple things that a user can be asked to try again on. Use more throws and try/catch blocks; note that custom exceptions may be more intuitive for understanding than use of detailed messages communicated through generic expections. Realistically, each exception should be reviewed individually to assess desired outcome from user/developer standpoint.

Review Code Management

Code organization

- Use of version control

  Review: The current Google Code + SubClipse combination is adequate for the project.

  Action: Make sure everything is well-documented for developers.

- Package structure

  Review: Package naming not consistent with ca.water.gov; too many classes, too few packages make the code difficult to understand.

  Actions: Package naming improved; reorganized classes into smaller, more logically organized packages (in progress).

- Configuration management

  Review: Considered what was needed to run the program, found that some things are not external and easily reconfigurable. One reviewer attempted to compile and test in an OS-X environment, and encountered difficulties.

  Action: At a minimum, clearly document requirements for development and target platforms, such as the (currently) unstated requirement that the product run in a Windows environment – which is necessary because the simulation tool invoked by the GUI only runs in a Windows environment. The requirements should then be reviewed to identify implementation aspects that are currently hard-coded but that can logically be externalized into a configuration file or files.

  Consistency of path treatment should be checked and documented as well. Paths should be relative paths throughout the code (for the user) and throughout the project (for the developer). Consider privileges, file system structure (separators).

  Question: Should users be able to run the GUI in any Java-friendly environment, and then send scenarios to someone else's Windows machine to evaluate in batch mode? Consider OS-X v. Win.

Testing

- Unit, integration and functional testing process

Review: Formal testing is not implemented in the project.

Action: Partial JUnit tests have been built for selected portions of the code (DSSGrabber, FileAction.setupAndRun). All new code should be developed with JUnit testing, and testing added opportunistically to other key code sections – which will also result in refactoring and general improvement in quality and maintainability of code.

Build

- Build process

  Review: No formal build process, though nature of project suggests that simple manual Eclipse build is OK; multiple versions of some external libraries – e.g. jFreeChart?

  Action: Implement formal build process – simple ant script that brings in version numbers for code and for GUI.xml; explain/resolve why multiple versions of JFreeChart are in the project.

- Use of continuous integration

  Review: Continuous integration is not currently relevant to this project, but future development process may make it so.

  Action: Will need to revisit an automated build process (including tests) as number of active developers increase, and discuss how to define the version for deployment.

Deployment

- Deployment process

  Review: No formal deployment process, responsibilities shared unclearly between DWR, Reclamation, contractor.

  Action: Discuss and document responsibilities, including steps to define deployment version of CalLite GUI vs. data files vs. simulation model vs. installation package. What is the delivery format - .jar, .zip, .exe? Actions taken here may have implications for continuous integration.

Review Code Clarity

Style standards

- Determine style standards followed

  Review: Formatting and naming standards not applied.

  Action: Standard style agreed on and implemented in Eclipse. Renaming to match Java standards to be done.

Documentation

Review: Generally, project documentation is inadequate

- Architecture level

  Review: Need a high-level architecture document.

Action: Update draft HLA document for developers, supplementing with package/class structure diagram and description, description of key external libraries (jFreeChart, jxBrowser, Swixml, …), and target environment for user (also ties to configuration management)

- Code level

  Review: Code-level documentation is inadequate.

  Action: Add JavaDocs documentation for all classes and for key methods

- Process management level

  Review: Development process is mostly ad hoc and not documented.

  Action: Document key parts of development process (and gently enforce) – tools (coding style/standards), roles, version control guidelines, testing, build, deployment

- Installation level

  Review: Installation process and documentation has to date been a client (BoR/DWR) responsibility.

  Action: Coordinate explicitly with staff responsible for installation process to identify necessary information to supply to support development of installation package.

Summary of Actions

This section summarizes the actions identified in the review process and presents suggested milestones.

1. GUI Development Team Coordination
   a. The GUI Development Team will use a consistent development environment, including platform (Eclipse Juno) and SVN (Subclipse), and use uniform formatting standards. (Goal: COMPLETE)
   b. The GUI Development Team will discuss and agree on a build system that can be used by all and accounts for multiple versions of referenced libraries. (Goal: By 1/18, LimnoTech will have a draft ANT build system in place for testing and review; by 1/25, the Development Team will use the system for weekly functional builds).
   c. The GUI Development Team will hold meetings every Friday to review progress, demonstrate updated functionality, discuss, prioritize, and assign issues, and agree on activities for the upcoming week. Attendance by at least one staff member from Reclamation, DWR, and LimnoTech expected (Goal: 100% representation at each of the 14 meetings scheduled from 1/18 through 4/19.)
   d. The GUI Development Team will build and release often. (Goal: at least six formal releases of the GUI to the Model Development Team – approximately biweekly in February, March and April - 2/1, 2/15, 3/1, 3/15, 3/29, 4/12)
   e. The GUI Development Team will discuss and reach consensus on branching and merging strategy. (Goal: By 1/18, publish guidelines in code wiki)
   f. Goal: all activity to be guided by Issues list on code site. Metric TBD.

g. The GUI Development Team will coordinate with the Model Development Team, particularly to define the requirements for a deployable installation package. (Goal: define basic requirements by 1/25, and report at 1/30 Model Development Team meeting; work with Model Development Team to define and develop one-click build of entire installation package, schedule TBD)

   (Goal: review progress monthly – 1/25, 2/22, 3/22 – and report at 1/30, 2/27, and 3/27 Model Development Team meetings)

2. Documentation
   a. Goal: A high-level architecture description satisfactory to Reclamation and DWR will be prepared by 1/18.
   b. Goal: 100% of existing classes will have Javadoc headers by 1/25.
   c. Goal: 100% of significant members of important existing classes – identified by consensus during the development of high-level architecture documentation and during ongoing review of code deodorization – will have Javadoc headers by 3/29)
   d. Goal: 100% of all new classes and all new non-trivial class members will have Javadoc headers.
3. Practices
   a. Goal: 100% of new code is to be testable with unit tests, documented with Javadoc, and reviewed.
   b. Goal: By 3/29, significant members of important existing classes identified as key code sections will be restructured – particularly through refactoring to reduce coupling, increase cohesiveness, improve performance (by using appropriate Java constructs, algorithms and libraries) and to support effective testing. Intent is to achieve >50% testing coverage of LOC.
   c. Goal: complete review and restructuring of FileAction.setupAndRun method and DataFileTableModel class by 1/25 (prior to implementation of new capabilities).
4. Suggested working arrangement – Kevin and Mike shadow Dan and Tad for larger issues/new development, but look for items that Mike or Kevin can lead on with Dan/Tad input. Nicky, Jun and Idrissa provide oversight and review.

Attachments

- High-level architecture document
- Issue list

CALGUI High-Level Architecture.

The CalLite GUI (CALGUI) is intended to improve accessibility and usability of the CalLite screening model for planning and management of the State Water Project and Central Valley Project in California. Figure 1 presents the initial conceptual view of CALGUI.

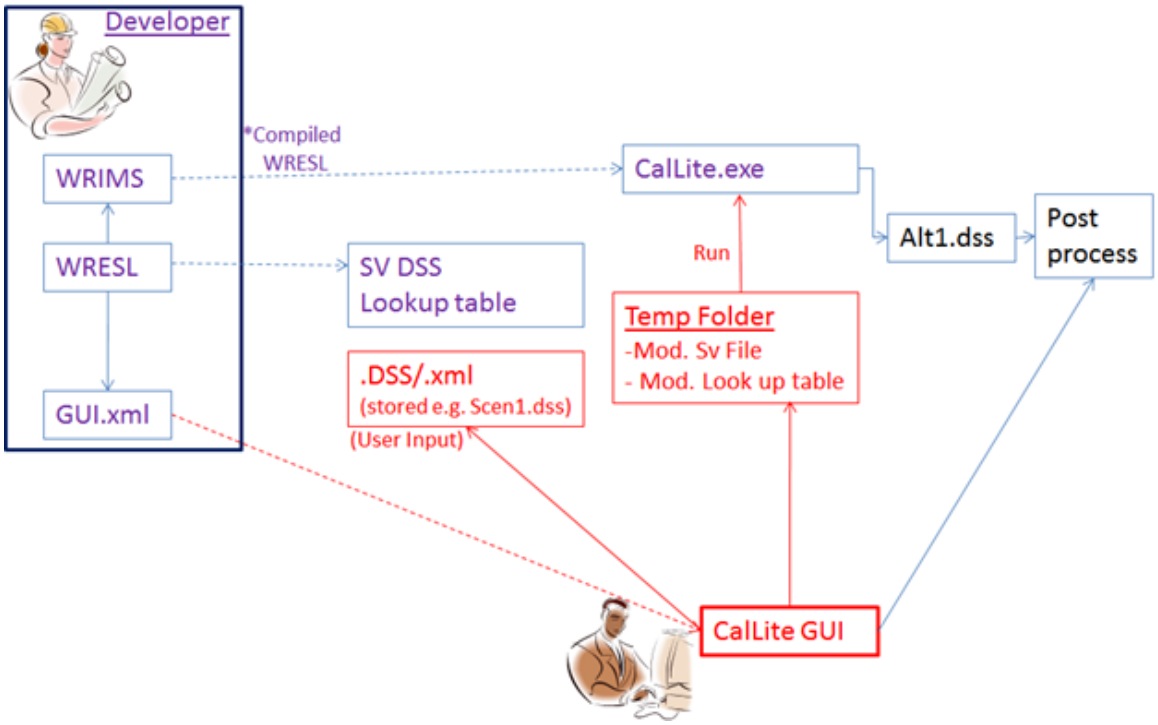

Figure 1. Initial conceptual design for CALGUI.

Key Functionality

The current CALGUI implementation (revision 328, 2/10/2012) implements the conceptual design with three key functionalities:

- On invocation, CALGUI builds an externally-reconfigurable GUI using GUI.xml and GUI_Links*.table files.
- The GUI is used by end users to create and run CalLite simulations ("scenarios") that consist of an ensemble of .dss (binary) and .table and .sty (ASCII) files. In some required .table and .sty files, particular values depend on user settings in the GUI, while the inclusion or exclusion of other .table files and of all .dss files are controlled by other GUI settings. Scenarios are defined by (and saves as) the state of the GUI.
- The GUI is also used to view or compare simulation results from CalLite scenarios.

High-Level Architecture

As shown in Figure 2, the major data flows in CALGUI parallel the functionality:

1. The files GUI.xml (defining content and layout of the UI), GUI_Links2.table (linking controls defined in GUI.xml to particular values in CalLite input files), GUI_Links3.table (describing result chart/table contents and layout) and GUI_Links4.table (identifying particular files to be included in a scenario based on GUI control settings) are read in by the main program.
2. A set of input files for the CalLite model is generated on demand by CALGUI. The input files are copied from a default set of input into a scenario subdirectory; some values in the input files will be modified according to control states in the GUI, while other files will be excluded or included according to control states. The overall state of the GUI is saved as a "scenario definition".
3. Once a scenario is evaluated by CalLite, its results can be viewed and compared to results from other scenarios using the various result dashboards (tabbed panes) in the GUI. Viewing of results requires the retrieval and transformation of time series data from HEC Data Storage System (DSS) binary files. A limited capability is provided to build and store collections of display options for ease of repeated use with different scenarios.

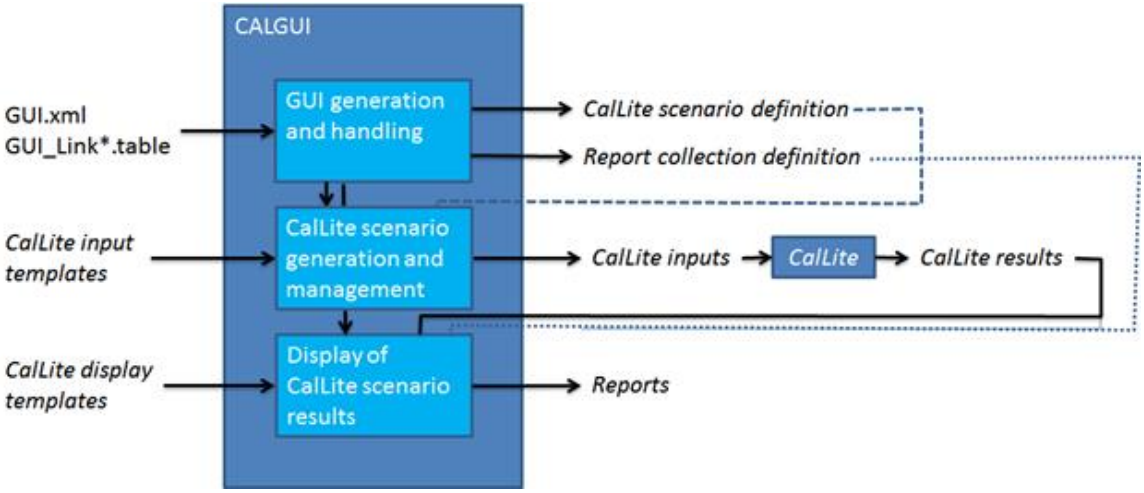

Figure 2. Draft diagram of high-level architecture

Location of Key Functionality

The key purpose of the CalLite GUI is to create, run, view, and manage scenario inputs and results. The scenario inputs are read from file into a Swing GUI implemented with the swixml library. Some specific notes

- The GUI is defined by:
  - External XML file that lays out GUI – read into MainMenu.swix (where in code?)
  - GUI_Link*.table – links GUI elements to entries in CalLite .table files.
  - Dashboard-specific special behaviors (where in code?) (example?)
- FileAction.setupAndRun – method creates a scenario from model embedded in GUI
  - Creates directory for run
  - Copies default input files to run directory

  - Writes select dashboard settings into *.table files according to GUI/model settings.
  - Creates appropriate study.sty files
  - Writes appropriate data tables into *.table files according to GUI/model settings
  - Other types of specific behaviors (e.g. which set of demand tables are copied in)

- DataFileTableModel – Class to handle data tables inside of GUI. Keeps track of user-defined and default table entries.

Current issues (code.google.com/CalLiteGUI – updated 1/9/2013)

ID	Summary
1	We haven't yet fully considered versioning
7	CalLite release feedback 1
8	Use SimpleFileFilter instead of all the XXXFileFilter classes.
10	Add exceptions to DSSGrabber
11	Write complete tests for DSSGrabber
12	DSSGrabber/MainMenu code organization
13	Change console output to logfile in DSSGrabber
14	Replace DSSGrabber NullPointerException with application-specific exception
17	Modify FileAction.setupAndRun to read settings from passed scenario file instead of SwingEngine
18	Put FileAction actions into separate methods?
19	Split GUIUtils into FileUtils and GUIUtils
20	return void from methods that are changing state of argument
21	close file when done reading in a finally block
22	No check done on file actions may cause app failure without appropriate messages
23	Externalize strings that contain values
24	Move data values from FacitlitesSetup into external file
25	Unused class FileListModel
26	Duplicate DSSFilter class?
27	Extract method to GUIUtils
28	Propagate exceptions to GUI
29	Use same log library for both GUI and wrims (same folder but different log file)?
30	Suggest changing DSSGrabber class variable access modifier cfs2TAFday
31	Duplicate monthToInt methods
32	Doesn't run nicely on Mac
33	Class-level javadoc needed
34	Tracking and handling CALGUI version
35	Multiple versions of .jars in paths?
36	Duplicate methods GUIUtils.monthToInt and GUIUtils.MonthStr2int
37	Address Run Scenario messaging issues.
38	Show the directory path in the Scenario Name box and the DSS File Name

39 Tooltips

40 If a scenario named Default.cls is not in the default scenarios directory, the GUI does not function properly.

41 Redesign regulations dashboard

42 Scenario Settings Dashboard

43 HydroClimate Dashboard

44 SV/Init Files

45 Facilities Dashboard - Shasta Enlargement

46 Facilities Dashboard - Sites Reservoir (North of Delta Offstream Storage - NODOS)

47 Facilities Dashboard - Los Vaqueros Enlargement

48 Facilities Dashboard - Temperance Flat Reservoir (Upper San Joaquin Storage Investigation)

49 Facilities Dashboard - Sacramento Valley Conjunctive Use

50 Facilities Dashboard - Isolated Facility

51 Facilities Dashboard - Banks Pumping Plant Capacity

52 Facilities Dashboard - Habitat Restoration Options

53 Quick Results Tab

54 Operations Dashboard - new options will be added to CVP/SWP allocation methods

55 Scenario Settings Dashboard

56 Custom Results Dashboard - variable selection

57 Custom Results Dashboard - SVARS and DVARS

58 Custom Results Dashboard - enable displays

59 Custom Results Dashboard - searching for strings

60 Custom Results Dashboard - MTS and DTS

61 Improve graphic display options - colors and styles

62 Improve graphic display options - scatter plot

63 Improve graphic display options - time-series

64 Improve graphic display options - box and whiskers

65 Improve graphic display options - double click

66 Improve graphic display options - GUI_links3.table

67 Improve graphic display options - HEC Library elements?

68 Map View Dashboard

69 Enable resizing of main CalLite GUI screen

70 Modify output dashboards to always have the left-hand pane for the the Quick Results

71 Auto load scenario DV file

72 Message box icons

73 Tab delimited file editing

74 Timestamp checks to .cls files

75 Add validity checking for GUI_Links tables at run time

76 Allow multiple selections in file selection dialogue for loading output files.

77 Splash screen

78 Explore HEC DSS file use for scenario files

79 Improve error handling

80 Implement documentation