Software Design Description

for

ArcLight Automation Testing

Contract No.: xxxxxxxx

C/SDRL No.: xxxxxxxx (if applicable)

Document No.: xxxxxxx, Rev. 007

Agency/Customer Number: xxx (if applicable)

26 February 2016

Prepared for:

**Customer**

**Address**

**City, State Zip Code**

Prepared by:



**ViaSat Inc.**

**6155 El Camino Real**

**Carlsbad, California 92009**

Table 1 - Revision Record

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Rationale** | **Release Date** | **Affected Pages** |
| Rev 001 | Preliminary Release focusing on High-Level Design | 24 April 2015 | All |
| Rev 002 | Release for Review | 22 May 2015 | All |
| Rev 003 | Intermediate update with on-going as-built changes | 06 July 2015 | All |
| Rev 004 | Update with as-built changes. Added Coordinates designs; Appendices with Server/Hub configurations | 31 Aug 2015 | All |
| Rev 005 | Update with as-built updates for Login/Status Tests | 31 Dec 2015 | All |
| Rev 006 | Minor additions for NLG Post Processing | 12 Jan 2016 | All |
| Rev 007 | Updates for ReDa progress | 26 Feb 2016 | All |

Table of Contents

[1 Scope 1](#_Toc444246704)

[1.1 Identification 1](#_Toc444246705)

[1.2 System Overview 1](#_Toc444246706)

[1.3 Document Overview 2](#_Toc444246707)

[2 Referenced documents 3](#_Toc444246708)

[2.1 Government/Customer Documents 3](#_Toc444246709)

[2.2 Commercial Standards 3](#_Toc444246710)

[2.3 Contractor Documents 3](#_Toc444246711)

[3 CSCI-wide design decisions 5](#_Toc444246712)

[3.1 Unresolved Design Decisions 6](#_Toc444246713)

[4 CSCI architectural design 7](#_Toc444246714)

[4.1 CSCI Components 7](#_Toc444246715)

[4.2 Concept of Execution 9](#_Toc444246716)

[4.2.1 NLG Data Analysis 10](#_Toc444246717)

[4.3 Interface Design 11](#_Toc444246718)

[4.3.1 Interface Identification and Diagrams 11](#_Toc444246719)

[4.3.2 Project-unique Identifier of Interface 11](#_Toc444246720)

[5 CSCI detailed design 35](#_Toc444246721)

[5.1 Apache 35](#_Toc444246722)

[5.2 AL Test GUI 35](#_Toc444246723)

[5.2.1 Main 36](#_Toc444246724)

[5.2.2 Test Case Launch 37](#_Toc444246725)

[5.2.3 Results 37](#_Toc444246726)

[5.2.4 Configuration 39](#_Toc444246727)

[5.2.5 Utilities 43](#_Toc444246728)

[5.3 AL Database 44](#_Toc444246729)

[5.3.1 ALDBCONN 44](#_Toc444246730)

[5.3.2 ALDBCONF 44](#_Toc444246731)

[5.4 AL Test Cases 46](#_Toc444246732)

[5.4.1 AL Automation Manager 46](#_Toc444246733)

[5.4.2 Map Bundles 46](#_Toc444246734)

[5.4.3 NLG 56](#_Toc444246735)

[5.4.4 NLG Main Pseudo-Code 57](#_Toc444246736)

[5.4.5 NLG Syslog Post Processing 59](#_Toc444246737)

[5.5 AL GUI 61](#_Toc444246738)

[5.6 ViaSat Selenium 61](#_Toc444246739)

[5.7 AL Devices 62](#_Toc444246740)

[5.8 GTAF Devices 63](#_Toc444246741)

[5.9 Results Database 63](#_Toc444246742)

[5.9.1 Pseudo-Code 63](#_Toc444246743)

[5.10 MySQL 64](#_Toc444246744)

[5.10.1 Pseudo-Code 64](#_Toc444246745)

[6 Requirements traceability 71](#_Toc444246746)

[7 Notes 72](#_Toc444246747)

[7.1 Definitions 72](#_Toc444246748)

[7.2 Acronyms 72](#_Toc444246749)

List of Tables

[Table 1 - Revision Record 1](#_Toc444248950)

List of Figures

[Figure 1 - System Overview Diagram 1](#_Toc444246750)

[Figure 2 - ReDa Results Example 5](#_Toc444246751)

[Figure 3 - phpMyAdmin Integrated with ReDa 6](#_Toc444246752)

[Figure 4 - ArcLight Automation for Map Bundles System Architecture Overview 7](#_Toc444246753)

[Figure 5 - ArcLight CSCI Components Interaction 8](#_Toc444246754)

[Figure 6 - ArcLight Automation Trickle Test Use Case 10](#_Toc444246755)

[Figure 7 - Terminal Syslog Analysis 11](#_Toc444246756)

[Figure 8 - Terminal Status Analysis 11](#_Toc444246757)

[Figure 9 - Homepage 13](#_Toc444246758)

[Figure 10 - Test Launch Screen 13](#_Toc444246759)

[Figure 11 - Test Results Screen 14](#_Toc444246760)

[Figure 12 - Test Results Detail 14](#_Toc444246761)

[Figure 13 - Test Results Status Database Export 15](#_Toc444246762)

[Figure 14 - Configuration Screen 15](#_Toc444246763)

[Figure 15 - Automation Tuning Configuration Screen 16](#_Toc444246764)

[Figure 16 - Automation Test Utilities Screen 16](#_Toc444246765)

[Figure 17 - Database Tables 34](#_Toc444246766)

[Figure 18- NLG Poller Architecture 57](#_Toc444246767)

# Scope

## Identification

This document is the Software Design Description (SDD) for the Test Automation Subsystem Software Product Configuration Items (CSCI), product number [TBR], developed for the ArcLight program. This document describes the detailed design for the design specified in the ArcLight Automation Testing SSDD. The initial rollout of this automation tool supported Automation for Map Bundles Testing. Phase 2a provided automated data collection features for VMT Status at Terminal and NMSPM associated with Network Login (“NLG”) and other testing. The data collected, some of which is post-processed, is organized into CSV files that may be imported into Excel and used for analysis. Phase 2b provided initial connectivity with the ReDa webpages and underlying MySQL support.

This document is maintained in Perforce location //Arclight/ArcLight/AcceptanceTest /Automation/ MapTesting/Docs/AL\_Automation\_SDD.docx

## System Overview

The Test Automation Subsystem will control Test Equipment and the ArcLight Unit Under Test to provide integrated automated/repeatable regression testing for Map Bundles, specifically Trickle Download of Map Bundles Verification, Map Bundle Upload via Terminal Webpage Verification, Precedence Verification of Map Bundles and Forward Link Lock Verification of Map Bundles. Additional test suites will support data collection for Network Login and other testing.

The Test Automation Subsystem will support a simple “LAMP” (Linux, Apache, MySQL, PHP/Python) web-stack for external Browser based test launch management (very basic) interface and test results review.

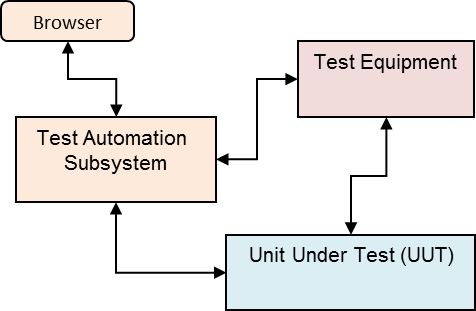


Figure - System Overview Diagram

## Document Overview

This SDD has been prepared using Data Item Description (DID) DI-IPSC-81435A (REF-21-01) as guidance. Each section is outlined below:

Section 1, *Scope*, of this document contains the introductory material regarding the purpose and scope of this document.

Section 2, *Referenced Documents*, provides a list of documents referenced within this SDD.

Section 3, *CSCI-wide Design Decision,* describes the design decisions that influenced the selection and design of the software components comprising the *[CSCI Abbreviation]* CSCI.

Section 4, *CSCI Architectural Design,* specifies the *[CSCI Abbreviation]* CSCI architecture which includes: identification of *[CSCI Abbreviation]* software components, the concept of execution between those components and the corresponding interface design for each software component.

Section 5, *CSCI Detailed Design,* specifies detailed design for each *[CSCI Abbreviation]* software component.

Section 6*, Requirements Traceability,* provides bi-directional traceability between the *[CSCI Abbreviation]* software components and associated requirements from the *[CSCI Name]* SRS.

Section 7*, Notes,* contains explanatory notes and a list of acronyms used within this SDD.

# Referenced documents

The following documents of the exact issue shown form a part of this document to the extent specified herein. In the event of inconsistencies between the documents referenced herein and the contents of this document, the contents of this document shall supersede.

## Government/Customer Documents

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier**  **(number or short mnemonic)** | **Document Number** | **Document Title** | **Date** |
| REF-21-01 | DI-IPSC-81435A | Software Design Description (SDD) | 15 Dec 1999 |
|  |  |  |  |

## Commercial Standards

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifier**  **(number or short mnemonic)** | **Document Number** | **Document Title** | **Date** |
| REF-22-01 |  | <https://dev.mysql.com/doc/refman/5.7/en/linux-installation-rpm.html> |  |
|  |  |  |  |

## Contractor Documents

|  |  |  |  |
| --- | --- | --- | --- |
| **Identifer**  **(number or short mnemonic)** | **Document Number** | **Document Title** | **Date** |
| REF-23-01 |  | //Arclight/ArcLight/AcceptanceTest/ Automation/ MapTesting/ AL\_Automation\_SSDD.pptx |  |
| REF-23-02 |  | //Arclight/ArcLight/AcceptanceTest/ Test\_Procedures/ AL1\_Mapbundles\_TestProcedure.docx |  |
| REF-23-03 |  | //Arclight/ArcLight/AcceptanceTest/ Automation/MapTesting/ Docs/AL\_Automation\_Sched.mpp |  |
| REF-23-04 |  | //Arclight/ArcLight/AcceptanceTest/ Automation/MapTesting/Docs/ AL\_Automation\_Bugs.xlsx |  |
| REF-23-05 |  | //Arclight/ArcLight/AcceptanceTest/ Automation/MapTesting/Docs/ AL\_Automation\_Integration\_Test\_Reports.pptx |  |
| REF-23-06 |  | //Arclight/ArcLight/ AcceptanceTest/ Test\_Plans/ AL1\_NLG\_NetworkLogin\_TestPlan.docx |  |
| REF-23-07 |  | //Arclight/ArcLight/AcceptanceTest/ Automation/MapTesting/Docs/ AL\_Automation\_User\_Man.pptx |  |

# CSCI-wide design decisions

The following design decisions guide the development and rollout of the CSCI’s:

* Implementation will be a phased approach with core functionalities rapidly developed and integrated according to the WBS (REF-23-03).
* Precedence processing will not be done as a group of N-maps all at once, but rather will be a group of N-results of individual precedence tests.
* Existing scripts for a ViaSat general purpose Selenium wrapper are available in python for use as-is, and should be used as the base class for deriving any additional functionality, i.e., new member functions.
* Existing scripts for ReDa come with many GUI based advantages such as Results presentation and phpMyAdmin which make porting the exiting code a net advantage to the project; see Figure 2 and Figure 3. The updates are minor, but a slight re-work to the GUI entry-point, i.e. index.html, is needed to support both ReDa and the ArcLight Test GUI.
* Existing ReDa library API’s are available in other scripting languages which also include other unneeded project specific extras, and these models can be used as a guide to adapt to ArcLight automation.
* Generating map coordinates on the Linux based server. Use license server to develop/compile and use MCR library for non-license runtime
* Generating kml file on the Linux based server; recompile for Linux

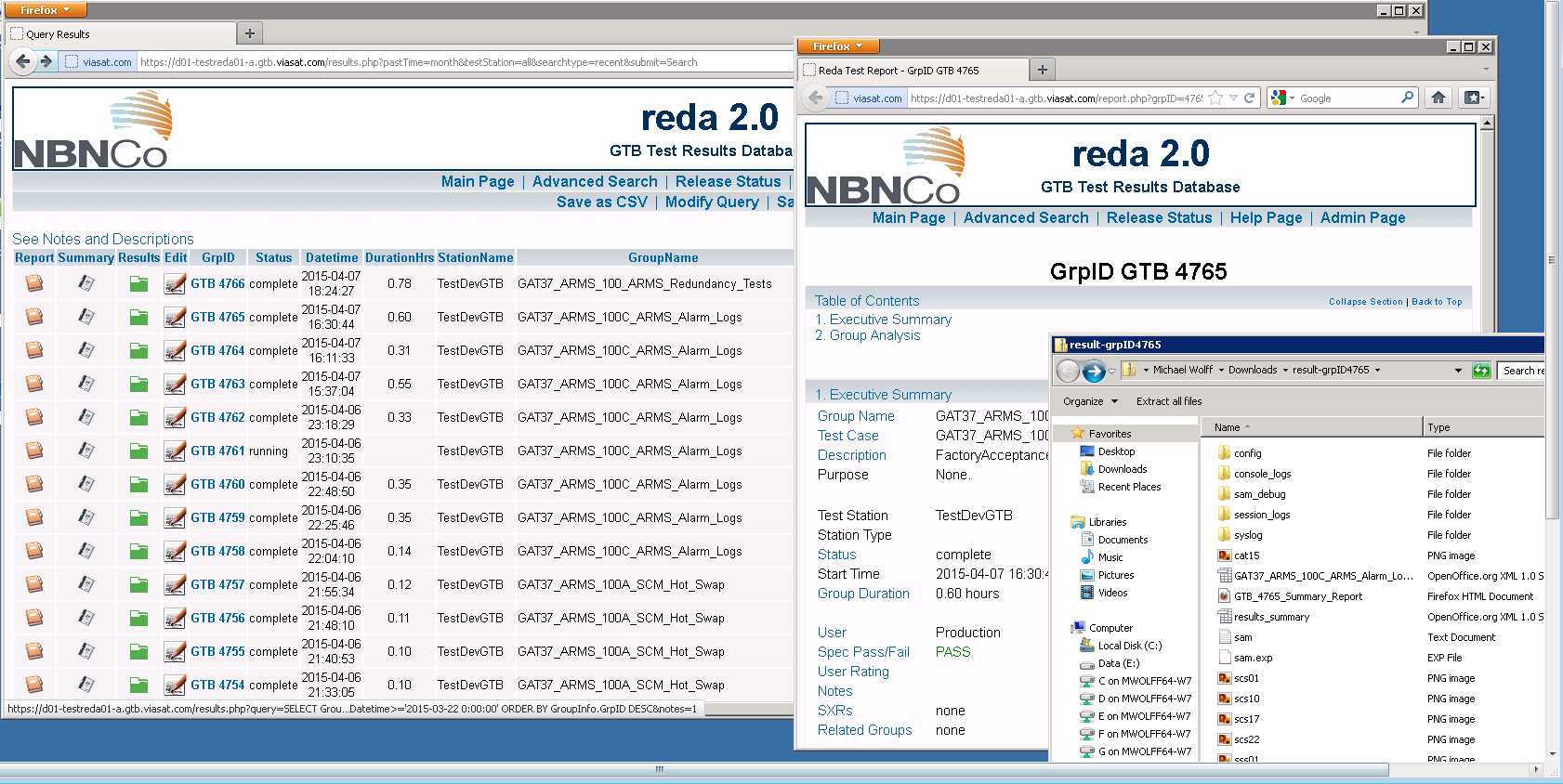


Figure - ReDa Results Example

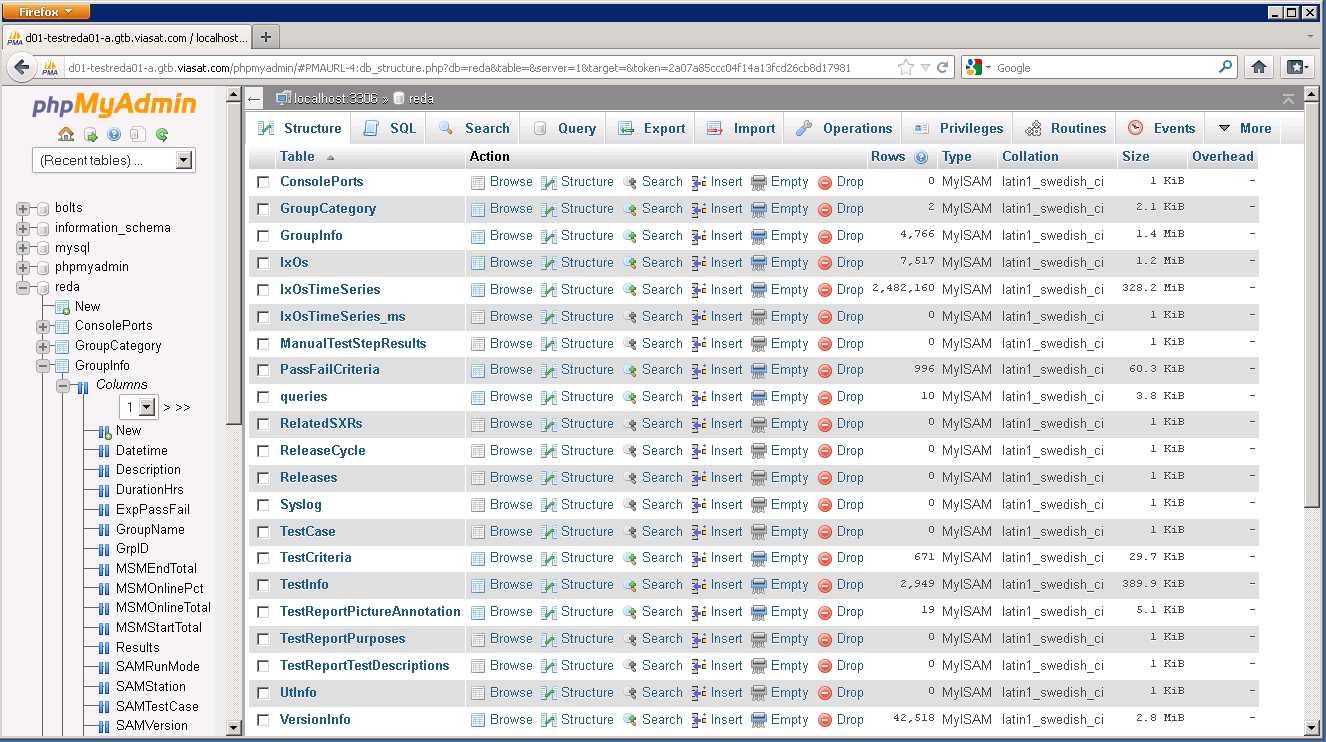


Figure - phpMyAdmin Integrated with ReDa

## Unresolved Design Decisions

The following design decisions need further investigation and resolution to complete the design (and/or implementation),

* Satellite (Beam) name may need to be a list of beams that have changed for the map; currently one beam per group. Can be tested in bulk by using group id range, e.g. 1-3,12,15,17-22.
* For ease in Apache implementation, currently the results are placed in /var/www/html/results/… and the code is located at /var/www/cgi-bin
* Merging SSH and Console packages for Terminal Commands (termcfg). The Terminal SSH interface is mandatory since the terminal console server is not a shared resource and was implemented in //Arclight/ArcLight/AcceptanceTest /Automation/MapTesting/ src/vsel/termLinkSSH.py which duplicates some code in //Arclight/ArcLight/AcceptanceTest /Automation/GTAF\_TIDE/ Scripts/utils/TermSerialCom.py

# CSCI architectural design

This section defines the software architecture at the component level and focuses on the inter-component interactions, i.e. the software high-level design.

Expanded from the system overview described in Section 1.2 , Figure 4 below shows a more in-depth system architecture containing COTS products from which the software architecture is derived. The Apache, MySQL, and Python/PHP elements, all running on Linux, provide the framework for external browser based command and control of the test system. The Python/PHP, Matlab [TBR], Selenium, Browser, Tcl/Expect elements provide the framework for control of the ArcLight system (unit under test).

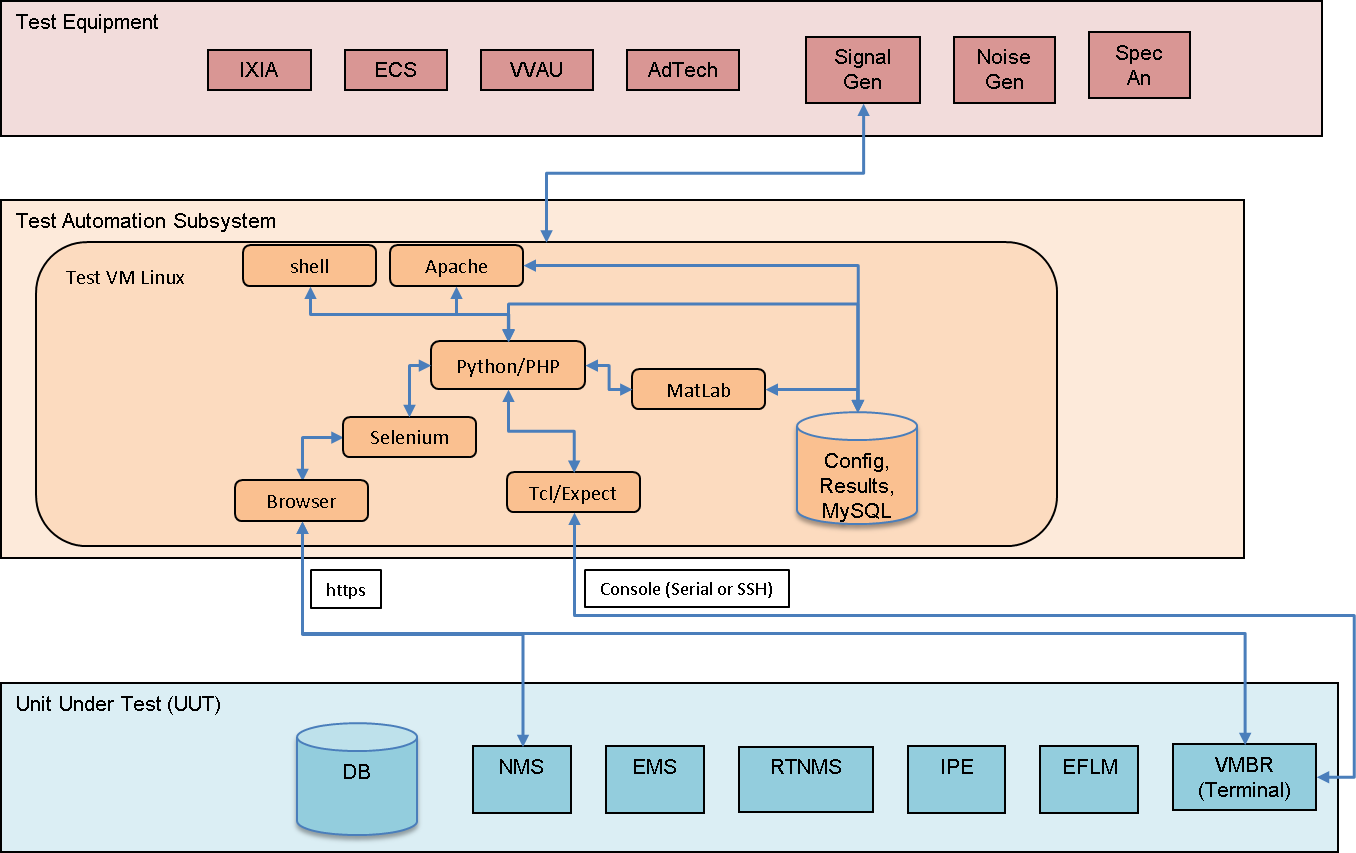


Figure - ArcLight Automation for Map Bundles System Architecture Overview

## CSCI Components

The CSCI components defined in the AL Automation SSDD (REF-23-01) and shown in

Figure 5 - ArcLight CSCI Components Interaction are summarized as follows

* ArcLight Test GUI manages simple set of html webpages for launching tests and getting results
* ArcLight Test Cases is a conceptual CSCI that contains:
  + ArcLight Automation Manager, which manages and controls test case execution that can be launched from AL Test GUI or Linux shell.
    - Common configuration of test database and files
    - Utilize API layers to manage devices during execution of test procedures
    - Extensible for additional test cases
    - Follows standardized results logging API (or ReDa)
  + Map Bundles Test Cases, which is a collection of scripts/objects for executing the map bundles test suite
    - mapBun Trickle
    - mapBun Terminal Upload
    - mapBun Precedence
    - mapBun FL Lock
  + Network Login/Terminal Status (NLG) Test cases
    - NMSPM VMT and Terminal Monitoring
    - NMS VMT Commands
    - Terminal and RTNMS syslog collection and post processing
  + Any other added Test Cases
* ArcLight GUI is a collection of scripts/objects for automating ArcLight NMS/EMS and Terminal GUI actions through the ViaSat Selenium CSCI or the Selenium Library.
* ViaSat Selenium is an existing collection of common scripts/objects for automating ViaSat browser based GUI actions.
* ArcLight Devices is a collection of scripts/objects for automating ArcLight devices through console interfaces, i.e. serial or ssh based CLI.
* GTAF Devices is a collection of scripts/objects for automating test equipment and providing a layer of common drivers for serial or ssh communications.
* ArcLight Database is a collection of scripts
* MySQL is the customization scripts for the COTS component.
* Results API (ReDa) is the Reda webpages which connect the operator interface with test results in the MySQL database and file artifacts.

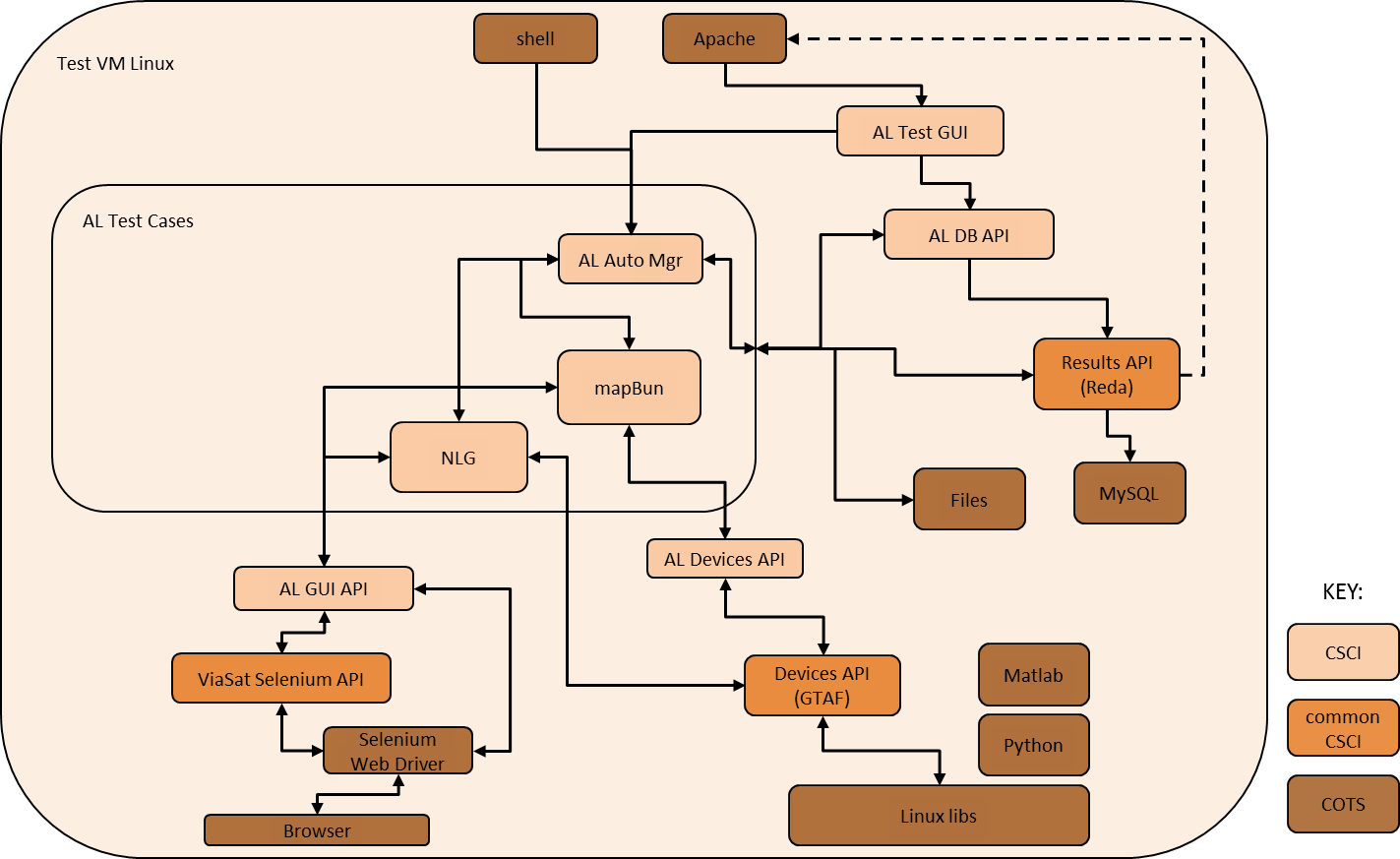


Figure - ArcLight CSCI Components Interaction

## Concept of Execution

Figure 5, shows the interaction of the CSCI components and the sequence diagram in Figure 6 shows the interaction/flow for the Trickle Test Use Case.

In the Trickle Test case example, as in most cases,

* The user at external browser creates the initial action with an HTTP GET (http interface) seen at the Apache server, thereby triggering a processing thread (Python/PHP script call).
* After initial configuration is set and stored in MySQL (sql interface), the test case (trickle) is started.
* The AL Test manager
  + invokes the mapbunConf API to configure the instance
  + invokes the mapbunTrickle API to execute the trickle of the bundle data associated with the instance
* mapbunTrickle uses ALGUI API’s that drive Selenium to
  + Login to the NMS
  + Upload the Map to NMS
  + Configure the download on NMS
    - At that point the UUT will download the bundle to the VMBR
  + Login to the Terminal
  + Get bundle version Status from the Terminal
    - Verify that status for pass/fail of the trickle
  + Invoke ALDB APIs that Store the results in MySQL
* The user views test results through the test results menus of the GUI

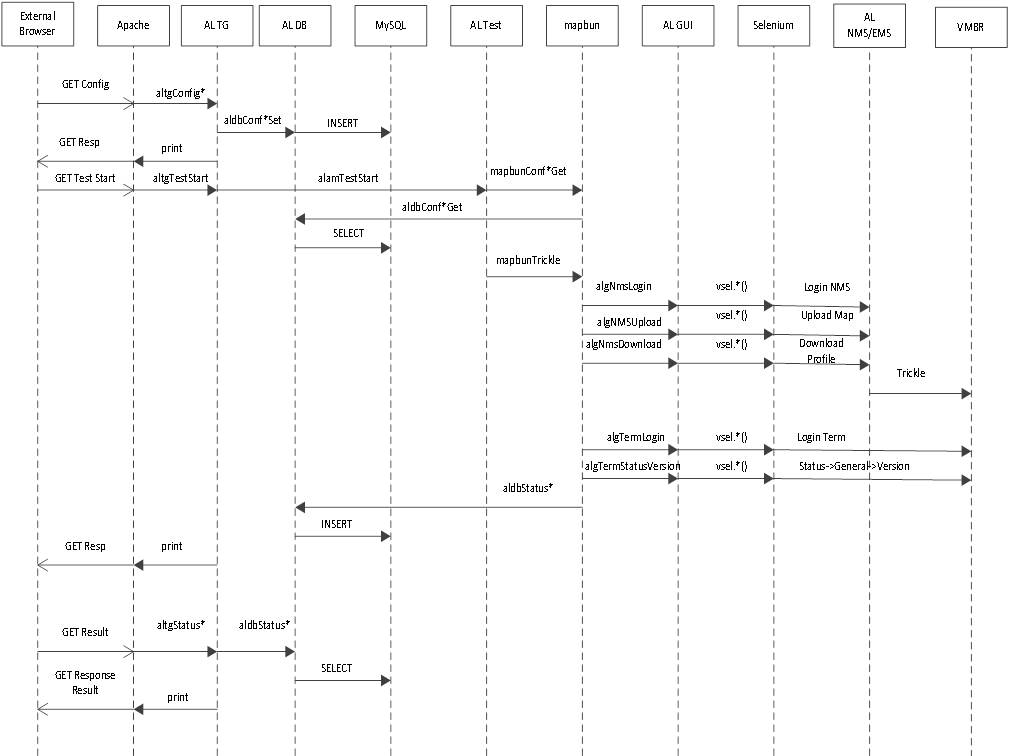


Figure - ArcLight Automation Trickle Test Use Case

The “Group Id” is a unique scalar identifier that associates a set Configuration Data and Test results. Once a test is run the group is locked.

### NLG Data Analysis

A key feature of NLG Data Collection is the ability to analyze post processed data in Excel. Upon completion of NLG collection capture, terminal and RTNMS syslogs are post-processed into condensed .csv files. Screen “grabs” of NMSPM and Terminal status are exported to .csv files too.

Figure 7 - Terminal Syslog Analysis shows a plot of the Terminal syslog .csv file generated by post-processor. Each column represents a binary event, and by plotting several columns over the same time period it is easy to interpret event “triggers” - and in this case reveal a CRL bug taking 2 or 3 times to login.

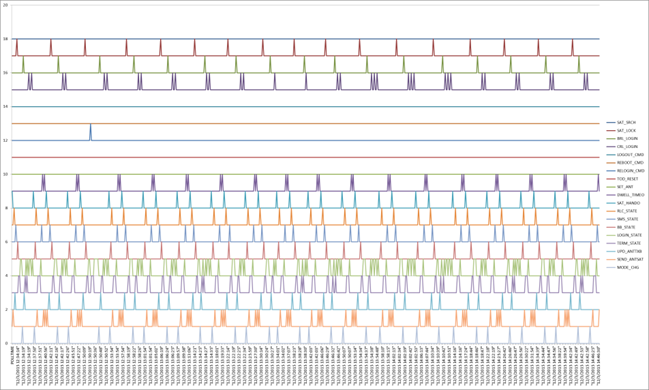


Figure - Terminal Syslog Analysis

Figure 8 - Terminal Status Analysis shows a plot of the RL DataRate displayed values in the Terminal Status .csv file exported from the collected status database. By corerelating that plot with the NMS RL Tx Bit Rate Chart captured result, it is simple to conclude that the the Terminal Display for RL Data Rate properly “trends” with the values reported to the NMSPM.

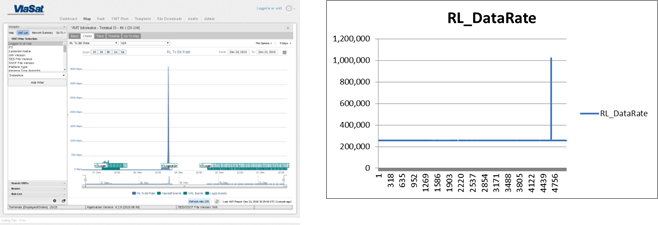


Figure - Terminal Status Analysis

## Interface Design

This section describes the interface characteristics of the software units.

### Interface Identification and Diagrams

The interfaces per CSCI are as follows:

* Apache - HTTP
* MySQL - SQL
* All other CSCI - API

### Project-unique Identifier of Interface

This section describes the API of each CSCI component.

#### Apache Interface

Apache will be configured to permit CGI script execution, but must also support location(s) compatible with ReDa; ReDa scripts will be deployed to the “DocumentRoot” or CGI location specified in httpd.conf, as applicable.

#### AL Test GUI Interface

The ArcLight Test GUI CSCI provides scripts responsible for managing the Human Machine Interface. This section describes the screens available and the URL requests that will be sent to the AL Test Linux device

##### Screens Interface

The following pages will be available through the Test GUI, i.e. the “human” interface

* Main (Homepage) - Contains buttons to the launch, results and configuration pages, see Figure 9 - Homepage
* Test Execution - Contains ability to launch test case(s), see Figure 10 - Test Launch Screen
  + Update single radio button selection, to more choices (checkboxes, radio, etc.)
* Test Results - Contains the executed test case status and results, see Figure 11 - Test Results Screen
  + [TBR] Results screen will be replaced with ReDa screen or deferred until ReDa can be ported.
  + Test Results Detail - Contains the test artifacts for the specific test case associated to the group, see Figure 12 - Test Results Detail
  + Status Database Export - Provides interface to export status database results into .csv file, see Figure 13 - Test Results Status Database Export.
* Test Configuration - Contains all parameters and will be saved in database with the exception of database parameters saved in a configuration file, see Figure 14 - Configuration Screen
  + Map Bundles Test Configuration Form - Contains following parameters needed for the map bundles testing
    - Map bundle zip file [from email]
    - Forward Link Profile Name
    - Satellite Name
  + NMS/EMS Connection - The IP, Port username and password needed for ArcLight NMS/EMS GUI access.
  + Terminal Connection - The IP, Port username, password, and Terminal Name needed for ArcLight Terminal GUI access, and ssh/serial, and Terminal console name (if serial), for console/CLI access
  + Tuning Parameters - Runtime parameters used by the various Test Automation algorithms, e.g. map precedence duration
* Test Utilities Menu - Contains testbed utilities
  + Map Bundles Test Report - Post processes results from specified group(s) to text file containing gleaned information needed to prepare QA test report

ReDa Menu - Launches the Reda php webpages

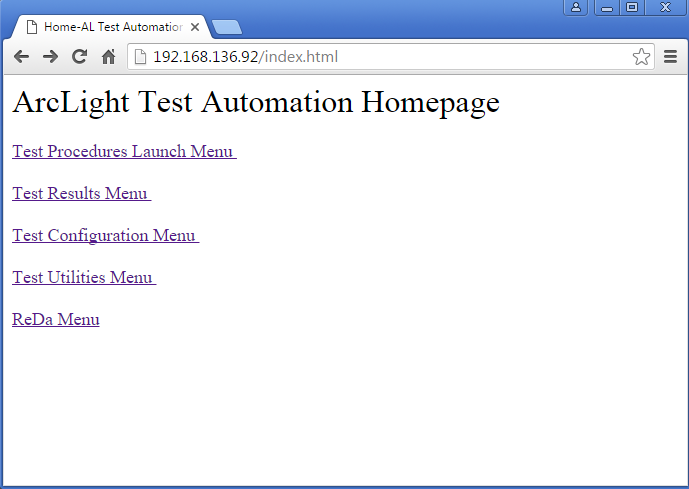


Figure - Homepage

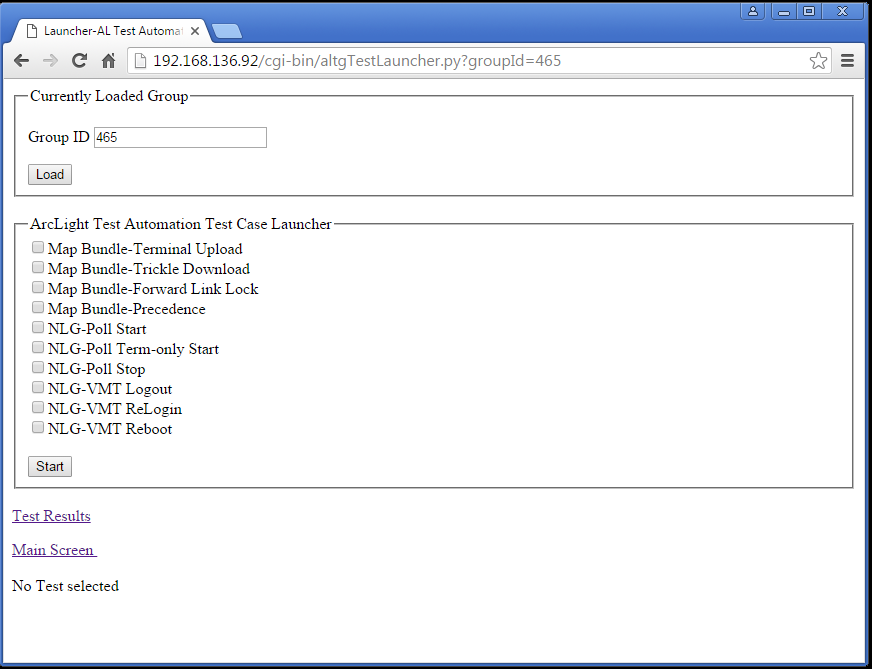


Figure - Test Launch Screen

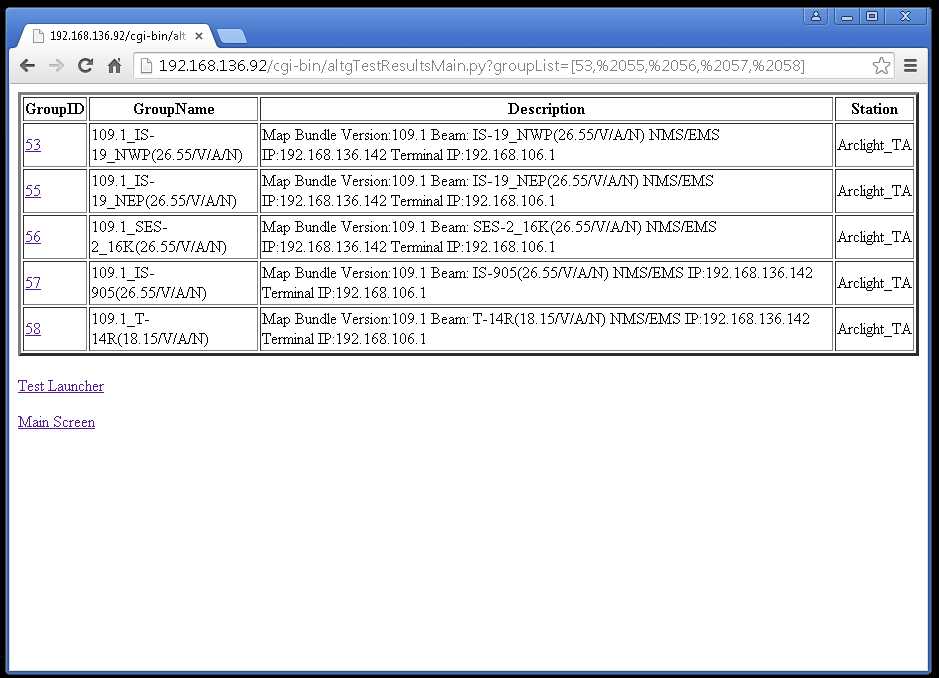


Figure - Test Results Screen

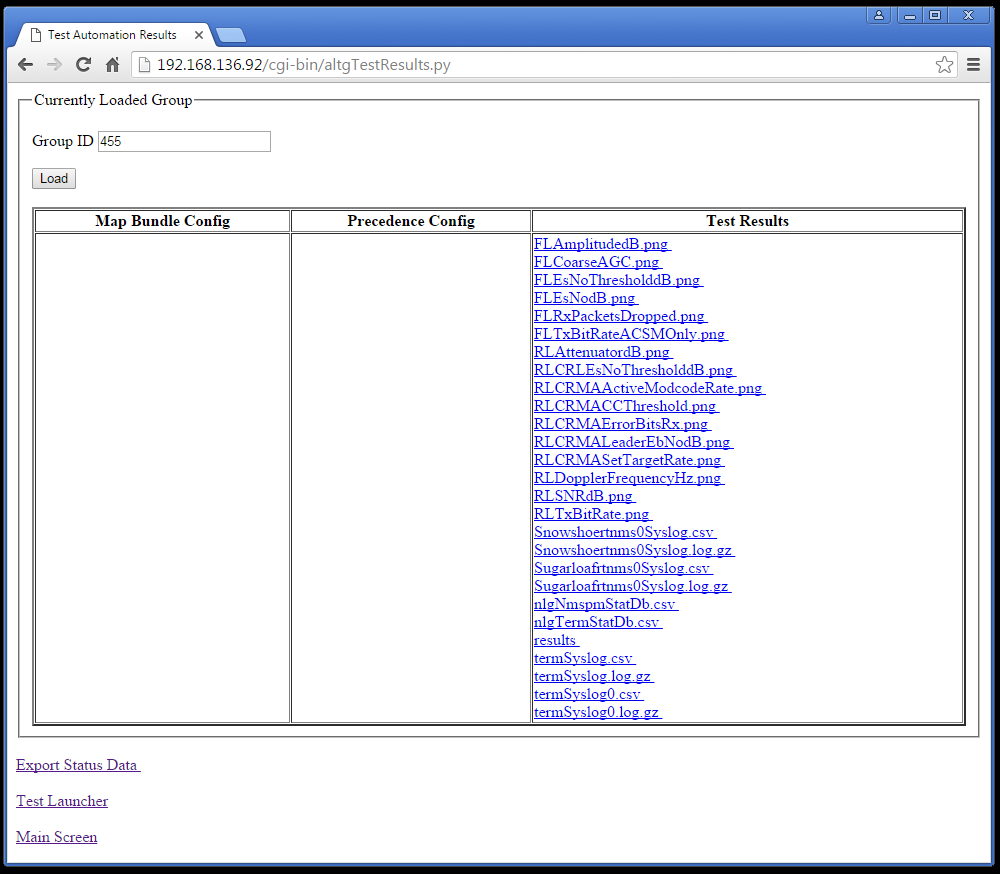


Figure 12 - Test Results Detail

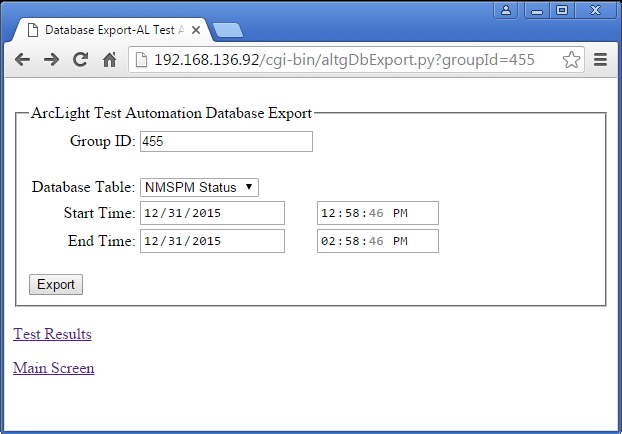


Figure 13 - Test Results Status Database Export

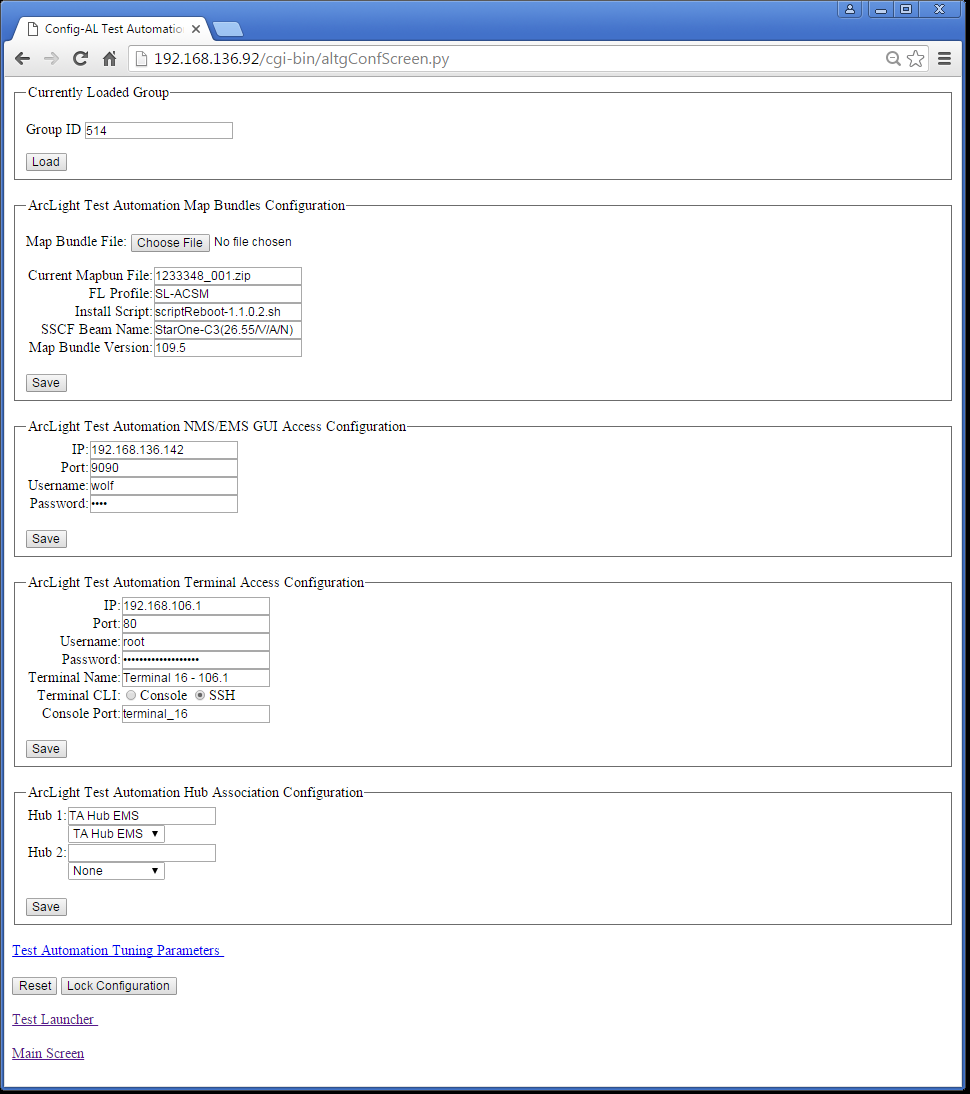


Figure - Configuration Screen

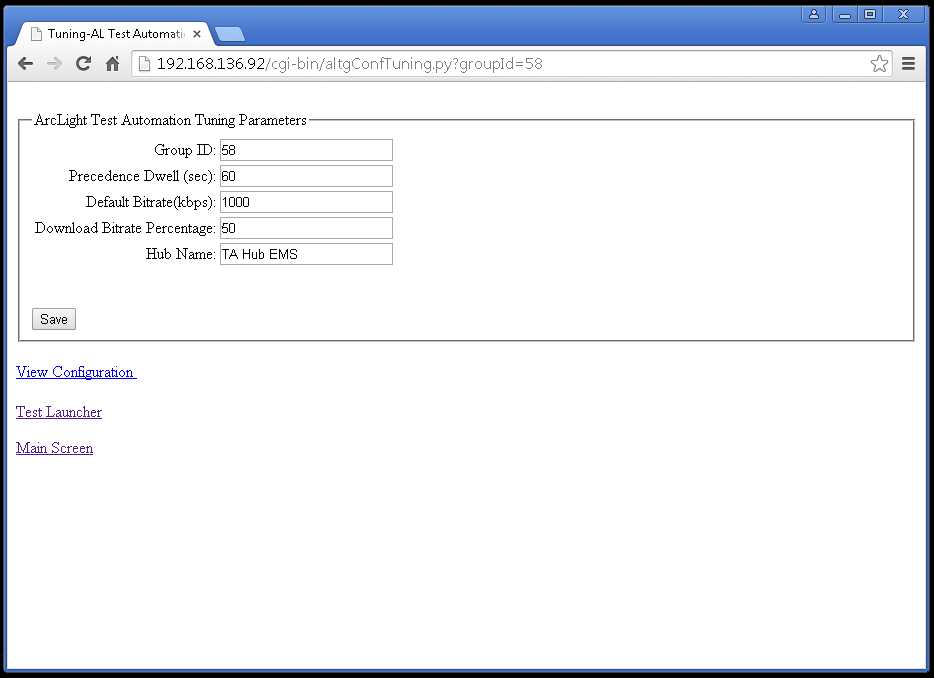


Figure - Automation Tuning Configuration Screen

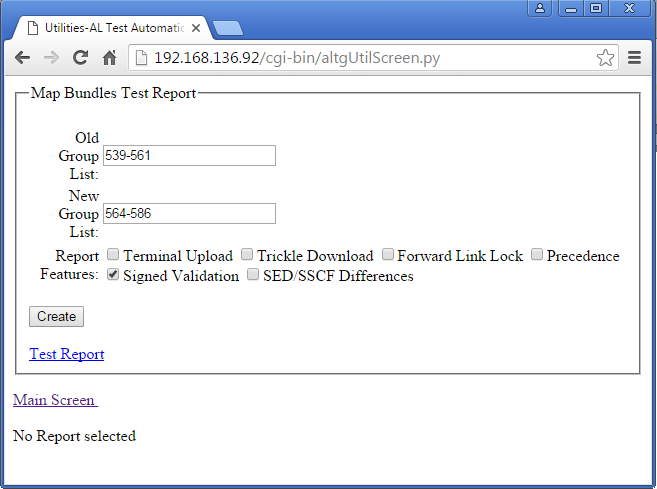


Figure 16 - Automation Test Utilities Screen

##### AL Test GUI Scripts Interface

This section defines the ArcLight Test GUI “web services” interface, including the expected parameters passed into action scripts from Apache. This definition provides the necessary ICD information needed for scripts as well as for HTML launched from the Test GUI Browser (encoded in the scripts). Unless otherwise specified, the access is a GET.

* http://altest.allab.viasat.com /index.html
  + http://altest.allab.viasat.com/cgi-bin/altgTest.py?
    - http://altest.allab.viasat.com/cgi-bin/ altgTestStart.py? Test=<TEST>
      * groupId=
      * where <TEST> is one of
        + mapTrickle
        + mapTerminal
        + mapPrecedence
        + mapFllock
        + nlgStartCollection
        + nlgStartCollectionTerm
        + nlgStopCollection
        + nlgVmtLogout
        + nlgVmtRelogin
        + nlgVmtReboot
        + nlgVmtBeamRedirect
  + http://altest.allab.viasat.com/cgi-bin/altgConfig.py?
    - http://altest.allab.viasat.com/ cgi-bin/altgConfigMap.py?
      * groupId=
      * $\_FILES
        + Map Bundles zip file
      * mapBundleVersion=
      * mapProfile=
      * mapScript=
      * mapSatelliteName=
    - http://altest.allab.viasat.com/cgi-bin/altgConfigNms.py?
      * groupId=
      * nmsIp=
      * nmsPort=
      * nmsUsern=
      * nmsPassw=
    - http://altest.allab.viasat.com/cgi-bin/altgConfigTerm.py?
      * groupId=
      * termIp=
      * termPort=
      * termUsern=
      * termPassw=
      * termName=
      * termAccessIsConsole=
      * termConsolePort=
    - http://altest.allab.viasat.com/cgi-bin/altgConfigTuning.py?
      * GroupID= (READ-ONLY)
      * precedenceDwell= (sec)
      * defaultBitrate= (kbps)
      * downloadBitratePercentage=
      * hubName=
  + http://altest.allab.viasat.com/cgi-bin/altgStatus.py?
    - http://altest.allab.viasat.com/cgi-bin/ altgStatusDetails.py?TestId=<VALUE>
  + <http://altest.allab.viasat.com/cgi-bin/altgDbExport.py>?
    - groupId=
    - dbTable=
    - startDate=
    - startTime=
    - endDate=
    - endTime=
  + http://altest.allab.viasat.com/cgi-bin/altgUtilScreen.py?
  + http://altest.allab.viasat.com/cgi-bin/reda/index.php?

#### AL DB Interface

Several parameters are needed to make a database connection and will be stored in a user config file, initially as a Python formatted import file that is intuitively a tag/value format, to overcome the “chicken or the egg” problem with storing database connection info in the database. The parameters are: database name, hostname, port number, username, and password, and will be managed by the aldbConn and aldbConnConfig package(s) with member functions for retrieving by scripts for database access.

Additionally, the ArcLight Database Model is managed by this package/layer. In general, the GUI sets configuration items that the AL Test Cases (Automation Manager) accesses (gets/retrieves) to execute tests according to the configuration.

##### Database Interface Manual Page

This user manual page describes the ArcLight Database API (ALDB).

SYNOPSIS

ALDB - ArcLight Database API

SYNTAX

from aldbConn IMPORT \*

from aldbConnConfig IMPORT \*

from aldbConf IMPORT \*

from aldbStatus IMPORT \*

def aldbConnParmsGet():

return { 'user':dbuser, 'password':dbpassw, 'host':hostname, 'port':portnum 'database':dbname}

class aldbConf(object):

def \_\_init\_\_(self, groupId):

def aldbConfNmsGuiSet(self, ip, port, usern, passw):

def aldbConfNmsGuiGet(self)

return(‘ip’:ip, ‘port’:port, ‘usern’:usern, ‘passw’:passw):

def aldbConfTermGuiSet(self, ip, port, usern, passw, name, iscli, consolePort):

def aldbConfTermGuiGet(self):

return(‘ip’:ip, ‘port’:port, ‘usern’:usern, ‘passw’:passw, ‘name’:name, ‘isCli’:iscli, ‘consolePort’:consolePort):

def aldbConfMapbunSet(self, filename, bundlever, profile, script, satname):

def aldbConfMapbunGet(self)

return(‘filename’:filen, ‘bundleVersionNum’:bundlever, ‘profile’:profile, ‘script’:script, ‘satName’:satname):

def aldbConfMapbunDetailedSet(self, signed, sedver, sscfver, rlcver, gdrmver, sedmd5, sscfmd5, rlcmd5, gdrmmd5, satname, satid, fltxfreq, flrxfreq, chiprate, antennatype):

def aldbConfMapbunDetailedGet(self):

return{ ‘signed’:signed,‘sedVersion’:sedver, ‘sscfVersion’:sscfver, ‘rlcVersion’:rlcver, ‘gdrmVersion’:gdrmver, ‘sedmd5’:sedmd5, ‘sscfmd5’:sscfmd5, ‘rlcmd5’:rlcmd5, ‘gdrmmd5’:gdrmmd5, ‘satName’:satname, ‘satId’:satid, ‘flTxFreq’:fltxfreq, ‘flRxFreq’:flRxfreq, ‘flChipRate’:flchiprate, ‘antennaType’: antennatype}

def aldbConfDirsGet(self):

return { ‘groupId’:groupId, ‘baseDir’:baseDir, ‘configDir’:configDir, ‘resultDir’:resultDir}:

def aldbConfHubsSet(self):

def aldbConfHubsGet(self):

def aldbConfRtnmsSet(self):

def aldbConfRtnmsGet(self):

def aldbConfNlgSet(self):

def aldbConfNlgGet(self):

def aldbStatusNmsPmBasicSet():

def aldbStatusNmsPmBasicGet():

def aldbStatusTerminalGenSet():

def aldbStatusTerminalGenGet():

def aldbStatusTerminalFLSet():

def aldbStatusTerminalFLGet():

def aldbStatusTerminalRLSet():

def aldbStatusTerminalRLGet():

def aldbStatusTerminalACUSet():

def aldbStatusTerminalACUGet():

def aldbExportToCsv():

DESCRIPTION

The ALDB API provides methods for managing the database for the ArcLight Automation device, by providing a common layer most notably accessed by the AL GUI and the AL Test Cases.

The aldbConn APIs provides methods for managing database connection data needed for connecting to the MySQL database even through a restart, by using data in a tag/value formatted file that the end user can edit.

aldbConnGetParms() returns the values needed to connect to the database and each return value may be retrieved using the result[] construct.

The aldbConnConfig.py file is the user customizable file with the following format:

dbname=altest

dbhostname=localhost

dbportnum=1234

dbusername=al1

dbpassw=Viasat123

aldbConfMapbunSet() sets map bundle configuration items; the filename is a complete path/filename of the map bundles zip file for the group Id associated with the class. The file path in this dataset points to the temporary location set by Apache; once the file is copied and extracted those filenames and locations will be placed into the configDir location, see aldbConfDirsSet/Get().

aldbConfMapbunGet() returns map bundle configuration items; the filename is a complete path/filename of the map bundles zip file for the group Id associated with the class.

aldbConfMapbunDetailedSet() and aldbConfMapbunDetailedGet() contain the calculated/extracted fields; determined and set by mapbundleConf().

aldbConfDirsGet() contain directory paths associated with the group Id. The directories are based on a formula and therefore there is no “set” routine; the aldb class is responsible for creating the directories for the groupId (in aldbConf.\_\_init()\_\_.

NOTES

The aldbconnConfig.py file may be setup differently on different deployments

Add these status items:

flName, flHub, flWaveForm, flRfBand

EXAMPLES

aldbConnGetParms()

usern = result[‘dbusername’]

SEE ALSO

N/A

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/aldbConn.py

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/aldbConnConfig.py

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/aldbConf.py

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/aldbStatus.py

#### AL Test Cases

The ArcLight Test Cases CSCI consists of the ArcLight Automation Manager, the Map Bundles test cases and all other automated test cases.

##### AL Auto Mgr Interface Manual Page

SYNOPSIS

ALAM - ArcLight Automation Manager API

SYNTAX

from alAutoMgr IMPORT \*

class alam(object):

def \_\_init\_\_(self):

def alamTestStart(self, testId, trickle=False, upload=False, fllock=False, precedence=False,nlgstart=False, nlgtermstart=False, nlgstop=False, logout=False, relogin=False, reboot=False):

DESCRIPTION

The ALAM package provides methods for configuring and starting automated tests.

alamTestStart() launches the tests which are True in an orderly (serial) fashion; the operator or ALTG is responsible for not mixing MAP and NLG tests or any incompatabilities otherwise.

NOTES

Create or Attach to Group Ids and Test Ids here

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/alAutoMgr.py

##### Map Bundles Interface Manual Page

SYNOPSIS

MAPBUN - Map Bundles API

SYNTAX

from mapBundles IMPORT \*

class mapbun(object):

def \_\_init\_\_(self, groupId):

def mapbunConf(self):

def mapbunTrickle(self, testId):

def mapbunUpload(self, testId):

def mapbunFLLock(self, testId):

def mapbunPrecedence(self, testId):

def mapbunPrecedenceVerify(self, computedFile, resultsFile, errorFile)

DESCRIPTION

The MAPBUN package contains common methods for setup and management of files, checksums and other test collateral needed by all map bundles tests, as well as specific methods for executing the Map Bundles test suites for trickle, upload, forward-link lock and precedence.

\_\_init\_\_()associates the manpbun instance to a specific results Group Id for accessing all configuration for the suite of tests to a common set of data.

mapbunConf() installs the configuration data (bundles zip file) to the config-results and builds out the extracted files and computes the md5 checksums. mapbunTrickle() executes the trickle download test using the map bundle associated with the group id of the class/instance. The NMS/EMS is configured and the map is verified at the terminal.

mapbunUpload() executes the map bundle upload test by uploading the map through the terminal’s local LAN port using the map bundle associated with the group id of the class/instance.

mapbunFLLock() executes the map bundle forward link lock test by setting the Hub Forward Link frequency and the terminal coordinates, then verifies that the terminal locks to the beam.

mapbunPrecedence()

mapbunPrecedenceVerify() returns True/False indicating the precedence test passed by comparing the theorectical lat,long, search list file with the result extracted from the terminal log.

NOTES

May need to break into smaller files, e.g. mapBunUtils.py, mapBunTrickle.py, etc.

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/ …/MapTesting/src/mapbun/mapbunMain.py

##### NLG Manual Page

SYNOPSIS

NLG - Network Login (Terminal Status) API

SYNTAX

from nlgMain IMPORT \*

class nlg(object):

def \_\_init\_\_(self,groupId, termonly=False):

def nlgNmspmOpen(self):

def nlgTermOpen(self):

def nlgNmspmClose(self):

def nlgTermClose(self):

def start(self):

def nlgNmspmStart(self):

def nlgTermStart(self):

def stop(self):

def nlgNmspmStop(self):

def nlgTermStop(self):

def nlgSigHandler(self, signum, stack):

def nlgNmspmPollStatus(self):

def nlgTermPollStatus(self):

def nlgNmspmTimerThread(self):

def nlgTermTimerThread(self):

def nlgNmsVmtCommand(self, command):

def nlgStartPoll(self):

def nlgStopPoll(self):

DESCRIPTION

The NLG package contains methods for starting and stopping polling threads for Terminal and NMSPM data-collection.

\_\_init\_\_()associates the NLG instance to a specific results Group Id for accessing all configuration for the suite of tests to a common set of data.

nlgNmspmClose() completes the NMSPM VMT status gathering by collecting the following results: FL/RL Chart screenshots, Rtnms syslog (including post processing into csv), exporting the NMSPM Status database into a .csv

nlgTermClose()completes the Terminal status gathering by collecting the following results: Terminal syslog (including post processing into csv), exporting the Terminal Status database into a .csv

NOTES

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/nlgMain.py

#### AL GUI Interface

This AL GUI manual page describes the ArcLight GUI user API which facilitates the ArcLight NMS/EMS and Terminal GUI control through the Selenium WebDriver Browser Automation.

##### AL GUI Interface Manual Page

SYNOPSIS

ALG - ArcLight GUI API

SYNTAX

from alGui IMPORT \*

class algElements(viasatSelenium):

\_\_init\_\_(self, resultsDir):

def algClickMenuTab(self, TAB):

def algClickSubMenuLink(self, LINK):

def algSelectTableRow(self, COLUMN\_NUM, TARGET):

def algLogin(self)

def algLogout(self)

def algScreenShot(self, filename)

def algSelectVMT(self, COLUMN\_NUM, TARGET):

def algStatusBasic1(self, delim=': '):

def algStatusBasic2(self, tagname, delim=':', valpath="/../../td[2]/div"):

class algNms(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

def algNmsLogin(self)

def algNmsLogout(self)

def algNmsDnldMapUpload(self, file, isSigned, antennaType)

def algNmsDnldMapProfile(self,profileName, isEnabled, defBitRate, isSigned, filename, fl, script, dnldBitRatePercent, vmtList)

def algNmsDnldMapStatus(self)

return{ ‘percentage’:perc, ‘errors’:numErr, ‘done’:done, ‘installed’:installed }

class algEms(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

def algEmsLogin(self)

def algEmsLogout(self)

def algEmsAcsmFlUpdate(self, profileName, hub, txIf, rxFreq, chip)

def algEmsFlEnable(self, hubName)

def algEmsFlDisable(self, hubName)

def algEmsAcsmFlSelectProfile(self, TARGET\_FL\_PROFILE):

def algEmsAcsmFlName(self, FL\_PROFILE):

def algEmsAcsmFlHub(self, HUB\_NAME):

def algEmsAcsmFlHubCenterFreq(self, HUB\_FL\_CENTER, TERM\_FL\_CENTER):

def algEmsAcsmFlRfBand(self, RF\_BAND):

def algEmsAcsmFlChipRate(self, FL\_CHIP\_RATE):

def algEmsAcsmFlRolloffFactor(self, ROLLOFF):

def algEmsAcsmFlPilot(self, PILOT):

def algEmsAcsmFlPbr(self, PBR\_IP\_1, PBR\_IP\_2, PBR\_IP\_3, PBR\_IP\_4):

def algEmsAcsmFlMcast(self, MCAST\_IP):

def algEmsAcsmFlState(self, ACSM\_ADAPT):

def algEmsAcsmFlModcode(self, MODCODE):

def algEmsAcsmFlExpandAdvanced(self):

class algTerm(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

def algTermLogin(self)

def algTermLogout(self)

def algTermMapUpload(self, sedfile, sscffile, rlcfile, gdrmfile)

def algTermConfVersion(self)

return{ ‘sedVersion’:sedver, ‘sscfVersion’:sscfver, ‘rlcVersion’:rlcver, ‘gdrmVersion’:gdrmver}

def algTermConfVerSave(self)

def algTermConfVerReboot(self)

def algTermStatusFLLock(self)

return{ ‘waveForm’:waveform, ‘flState’:flState, ‘dlFreq’:dlFreq, ‘esno’:esno, ‘amplitude’:amplitude, ‘agc’:agc,

‘modcod’:modcod, ‘modul’:modul, ‘lbandFreq’:lbandFreq, ‘lnbloFreq’:lnbloFreq, ‘rfFreqOff’:rxFreqOff }

def algTermStatusVersion(self)

return{ ‘sedVersion’:sedver, ‘sscfVersion’:sscfver, ‘rlcVersion’:rlcver, ‘gdrmVersion’:gdrmver}

def algTermGoToPage(self):

def algTermStatusGeneral(self):

def algTermStatusFL(self):

def algTermStatusRL(self):

def algTermStatusACU(self):

def algTermStatusTable(self,sections,prefixNames):

def algTermStatusTableToDict(self,keyLocator,valueLocator,prefixName):

class algNmspm(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

def algNmspmLogin(self)

def algNmspmLogout(self)

def algNmspmSearchVMT(self, searchText):

def algNmspmGetBasic(self):

def algNmspmChartScreenShots(self,chartPrefix):

def algNmspmChartScreenShot(self, fileName, chartName1=None, chartName2=None):

def algNmspmGetTrace(self, fileName):

def algNmspmGetFlightHistory(self, duration, rowNum=1):

def algNmspmFlightHistoryRow(self, rowNum=1):

def algNmspmGetRow(self, headerRowLocator, rowLocator):

DESCRIPTION

The ALG package provides APIs for automating operator actions on the ArcLight GUI for NMS, NMSPM, EMS and Terminal with the algNms, algEms and algTerm classes, respectively. All of those classes support login and logout member functions; the remainder functions are defined as follows:

algNmsDnldMapUpload(), algNmsDnldMap(), and algNmsDnldMapStatus()support the NMS GUI portions of the map bundle trickle test to upload the file, initiate the FL download and check the trickle status from NMS perspective, respectively.

algEmsUpdateACSMFwdLink() supports the EMS GUI portions of the FL Link Lock test for configuring the freq and chip rate values.

algTermStatusFLLock() supports the Terminal GUI portions of the FL Link Lock test for polling lock status and other forward link status as available on the Terminal GUI.

The NMSPM and TermStatus member functions are associated with status screen gathering and return python dictionaries indicating the status field name on the screen and the associated value as the key/value pair.

NOTES

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/alGui.py

#### ViaSat Selenium Interface

The ViaSat Selenium (VSEL, or VSELENIUM) layer abstracts the Selenium libraries to create general purpose GUI operations intuitive across all platforms, i.e. click a button, or select an item from a list. This interface can be used for helping to automate any ViaSat browser-based GUI product with Selenium.

VSEL is derived (copied) from the Surfbeam Automation Manager code-base, specifically from the file //Broadband/TestTools/SAM/core/lib/Selenium2.py

##### ViaSat Selenium Interface Manual Page

SYNOPSIS

VSEL (or VSELENIUM) - ViaSat Selenium API

SYNTAX

from viasatSelenium IMPORT \*

class vselenium(Selenium2):

def \_\_init\_\_(self, browserType, testResultsDir, pause=10):

# Fixes

def connectToServer(self, url, titleString=None):

def disconnectFromServer(self):

# Extensions:

def timestamp(self):

def clickButton(self, TEXT):

def clickCheckbox(self, def clickCheckbox(self, checkboxName, state):

def selectPulldownOption(self, pulldownName, pulldownOption):

def selectPulldownStartsWith(self, pulldownName, pulldownOption):

def selectPulldownContains(self, pulldownName, pulldownOption):

def chooseFile(self, nameLocator, file):

def scrollToText(self, text):

def connectToServer(self, url, titleString):

def disconnectFromServer(self):

def \_\_enter\_\_(self):

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

def clickImage(self, imageURL):

DESCRIPTION

The VSEL package provides general purpose GUI methods for locating, selecting, clicking browser objects as well as creating test artifacts by taking screenshots. This package can be used for helping to automate any ViaSat browser-based GUI product with Selenium.

This package inherits, “improves” and extends the Selenium2.py package from //Broadband baseline. Selenium2 was updated to support a variable “pause” timer rather than hardcode to “10”; all other updates are in made by VSEL in vselMain.py

\_\_init\_\_() sets the pause timer for this project as dictated by the upper layer package (in this case ALG)

clickButton() clicks a button with the specified text; caveat undeterministic if multiple buttons with same text on single screen

teardown() destroys the chromedriver session

clickCheckbox() clicks the check box with the associated label

selectPulldownOption() selects the option from the pulldown menu that exactly matches the pilldownOption

selectPulldownStartsWith() selects the option from the pulldown menu that has 'puldownOption' as ts prefix

selectPulldownContains() selects the option from the pulldown menu that is a substring match with 'pulldownOption'

chooseFile() selects specified file with the browser choose file button

scrollToText() scrolls the browser to desired text, presumably for snapshot

connectToServer() Allows title to be omitted, and therefore omits title assertion

NOTES

For ArcLight, use: Browsertype = ”chrome”, selectPulldownStartsWith()

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/viasatSelenium.py

#### AL Devices Interface

The ArcLight Devices Interface provides methods for controlling non-gui interfaces of ArcLight Equipment

##### Terminal Console Interface Manual Page

The Terminal Console Interface provides methods for executing ArcLight Terminal commands whether connected via the serial port or ssh.

NOTES:

* Currently, the existing library handles via serial port; as such, some of these classes should be moved to the GTAF devices layer where a serial or ssh transport path can be chosen independently.
* As a stop-gap, termLinkSSH.py was created to handle these functions over SSH

SYNOPSIS

TERMCON - ArcLight Terminal Console API

SYNTAX

from TermSerialCom IMPORT \*

class ConsolePort(abstract.FileDescriptor)

def \_\_init\_\_(self, protocol, console\_name, reactor):

def write(self, data):

def fileno(self):

def writeSomeData(self, data):

def doRead(self):

def connectionLost(self, reason):

class TerminalLink(basic.LineReceiver):

def \_\_init\_\_(self, port, log):

def transmit(self, data):

def sendCommand(self, cmd):

def resetWaitingStatus(self):

def rawDataReceived(self, data):

class Cmd

def addCallback(self, f):

def addErrback(self, f):

def addBoth(self, f):

def send(self):

def rxData(self, data):

def timeoutExpired(self):

def reRespSearch(self, regexp):

class Log

def setup(self,filename):

def openCapture(self, filename):

def openCSV(self, filename):

def write(self, data):

def writeRecord(self, datalist):

def close(self):

class Console(basic.LineReceiver)

def \_\_init\_\_(self, tl, listener = None):

def dataReceived(self, data):

class PwrCmd

def \_\_init\_\_(self, tl):

def pwrOn(self, waitTime):

def pwrOff(self, waitTime):

class TermCmd(object):

def \_\_init\_\_(self, tl):

def login(self, user, password):

def logout(self):

def checkVersions(self):

def checkTxrVersion(self):

def checkLoginStatus(self):

def uploadFile(self, ftpAddress, filename, targetName=None):

def changeHubLogin(self, HubName, waveform):

def setLatLong(self, lat, lon):

def changeTxrVersion(self, ftpAddress, imageFileName, versionName, postFlashCmd=None):

def renameFile(self, source, destination):

def rmFile(self, target):

def verifyFile(self, filename):

def getRLDataRate(self):

def getAttenuator(self):

def ctrlC(self):

def ctrlD(self):

def termcfg(self):

def termcfgVersion(self):

def termcfgStatus(self):

def termcfgRLConfig(self):

def getAttenLimit(self):

def showDate(self):

def reboot(self):

def reboot\_error(self):

def catMdlcini(self):

def removeMdlcDatabase(self):

def resetAcu(self):

def activateDemodulator(self):

def idleDemodulator(self):

def waitTxEnable(self):

def waitLogin(self):

def checkFileRxInit(self):

def checkFileRxFail(self):

def checkFileRxReschedule(self):

def waitFlashBegin(self):

def waitFlashComplete(self, postFlashCmd=None):

def dynamicFunctionCaller(self, inp):

def grepLog(self, target):

def execCmd(self,cmd,timeout=5):

def wait\_Idle40(self):

def termCfgStatusInit(self):

def refreshStatus(self):

def refreshAttenuator(self):

def getEth0Availibility(self):

def getEth0Speed(self, speed):

def getEth0Duplex(self, duplex):

def waitCmd(self, time):

def getEth0Ping(self, ipPing):

def waitConsoleTerm(self):

### Extensions

def setAntenna(self, antennaType)

DESCRIPTION

The TermCmd class encapsulates commands for controlling and checking status of the terminal. Typically when calling these functions, one should use the defer.inlineCallbacks decorator on the calling function. This decorator causes the calling function to stop execution at the yield statement and wait for the result of the expression in the yield statement before continuing. While waiting for the result, control is returned from the calling function. This allows the Twisted Python reactor to continue processing during the wait, which is essential because the reactor is in charge of handling reception of data over the serial connection (if the calling function blocked, then it would be unable to receive any responses due to the single-thread, single-process framework. To return a value from a function using the defer.inlineCallbacks generator, use the defer.returnValue function. This class should handle differences between terminal versions, such as image naming conventions, program name changes, login changes, and menu changes.

The PwrCmd class encapsulates commands for controlling the serial controlled power unit.

The TerminalLink class extends the Twisted basic.LineReceiver protocol for handling the details of sending and receiving data over a serial connection between a computer and an Arclight terminal. It holds a reference to the currently pending command so that received data can be directed to that Cmd object for accumulation in its response buffer.

The Cmd class encapsulates specific information about a command/response pair for passing to callbacks. Tracks transmit time of command, receives time of response, and the text of the response.

The Log class handles recording of data, writes terminal output to the screen and to a capture file, and writes data fields to a .csv file.

The Console class extends LineReceiver protocol to handle asynchronous console input over serial.

NOTES

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/GTAF\_TIDE/Scripts/utils/TermSerialCom.py

#### GTAF Devices Interface

The GTAF Devices CSCI is not being developed in the initial phase.

The following generic ssh class

SYNOPSIS

viasatSsh - ssh/scp utility

SYNTAX

from viasatSsh IMPORT \*

class viasatSsh

def \_\_init\_\_(self, host, usern, passw, log=None):

def getFile(self, srcfn, dstfn):

DESCRIPTION

The viasatSsh class supports provides generic scp and is specifically used for retrieving syslog files from terminals/rtnms

#### Coordinates Interface

The Legacy coordinates package, a.k.a find\_all\_coords.m. is a MATLAB package that provides an output file, coords\_v<bundle\_version>.txt. That file is used as one of the input files to the legacy map\_bundles.py (which calls SED\_SCCF\_bundletest.py) script

The legacy API invocation is via MatLab:

* find\_all\_coords(‘<sedFile>’, ‘<sscfFile>’, ‘,<satListFile>’, ‘<bundleVer>’);
  + Result is a file in the current working directory, coords\_v<bundleVer>.txt
    - Contains coordinates for every intersection of the beams in <satListFile> with any beam in <sedFile>
    - <sedFile> and <sscfFile> must not have the md5 checksum

The new and improved API, specified in the Manual Page below, will provide for a Linux based executable invoked from Python and/or shell scripts for automation.

##### Coordinates Interface Manual Page

SYNOPSIS

COORDS - Coordinates for Map Bundle package

SYNTAX

run\_findAllCoords.sh <mcr\_path> <sedFile> <sscfFile> <satListFile> <bundleVer> [<precByIdFile> <precByNameFile> <coordsFile> <beamPlotFile>]

run\_findAllCoords.sh <MATLAB\_PATH> <sedFile> <sscfFile> <satListFile> <bundleVer> [<precByIdFile> <precByNameFile> <coordsFile> <beamPlotFile>]

findAllCoords <sedFile> <sscfFile> <satListFile> <bundleVer> [<precByIdFile> <precByNameFile> <coordsFile> <beamPlotFile>]

run\_findAllCoords.sh <MATLAB\_PATH> <sedFile> <sscfFile> <satListFile> <bundleVer> [<precByIdFile> <precByNameFile> <coordsFile> <beamPlotFile>]

find\_all\_coords(‘<sedFile>’, ‘<sscfFile>’, ‘,<satListFile>’, ‘<bundleVer>’

[‘<precByIdFile>’, ‘<precByNameFile>’, ‘<coordsFile>’, ‘<beamPlotFile>’)]

DESCRIPTION

This package contains the API’s necessary to generate LAT/LONG coordinates for testing terminal precedence. The package consists of MATLAB core functions/programs to generate coordinate file(s) and precedence verification file(s).

run\_findAllCoords.sh is the linux executable file generated by the MATLAB compiler to launch the findAllCoords stand-alone executable. For automation, this file will be called by the mapbunCoordinates.sh script. If the optional parameters [<precByIdFile> <precByNameFile> <coordsFile> <beamPlotFile] are specified, then the find\_all\_coords v2 algorithm will be run, otherwise the find\_all\_coords legacy algorithm will be run. The <mcr\_path> is the location of the MATLAB runtime libraries, e.g. /usr/local/MATLAB/MATLAB\_Compiler\_Runtime/v83

findAllCoords is the linux executable file generated by the MATLAB compiler to execute the find\_all\_coords() program.

find\_all\_coords with none of the optional parameters is the legacy algorithm to create a ./coords\_v<bundleVersion>.txt file containing all intersecting points for the satellite beams specified in <satListFile> and outputs map\_v<bundleversion>.pdf. By specifying the optional parameters, the v2 algorithm is run to create a <coordsFile> file containing optimized intersecting points (points with most beams overlapping) for the satellite beams specified in <satListFile>

File Formats:

<coordsFile> - intersections of satellite beams with beams in <satListFile> (v1 has all; v2 has minimum set to cover all beams

Lat, Long

Lat, Long

…

<precByIdFile> - Precedence points by beam number in priority for each Lat/Long point (v2 only; used to verify Terminal follows proper precedence)

1,31.32,-89.76,1,6,22,18,20

1,32.12,-81.76,1,6,22,25,15,20

…

<precByNameFile> - Precedence points by beam Name in priority for each Lat/Long point (v2 only; used to verify Terminal follows proper precedence)

SES-2\_16K(26.55/Y/A/N), 31.32, -89.76, SES-2\_16K(26.55/Y/A/N), Horizons-1(26.55/Y/A/N), AMC-21\_3S(26.55/Y/A/N), SATMEX-5(13.2/Y/A/N), SES-4(26.55/Y/A/N)

SES-2\_16K(26.55/Y/A/N), 32.12, -81.76, SES-2\_16K(26.55/Y/A/N), Horizons-1(26.55/Y/A/N), AMC-21\_3S(26.55/Y/A/N), T-14R(18.15/Y/A/N), T-11N(19.95/Y/A/NMAOR/N), SES-4(26.55/Y/A/N)

…

<sedFile> and <sscfFile> must not have the md5 checksum

<satList> name of beams to generate coordinates

SES-2\_16K(26.55/Y/A/N)

<beamPlotFile> output file of contours and coordinate points of map bundle

JPEG format currently

NOTES

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/Scripts/Maps/find\_coords/

##### Sed2Kml Interface

The sed2kml MATLAB program located at //Arclight/ArcLight/Engineering/deployment/MapFiles/MapTools/KmlConversion/sed2kml.m, has been built for Linux execution with the MATLAB Compiler Runtime (MCR). The file is invoked as

SYNOPSIS

SED2KML - Generate a Google Earth KML readable file from the Map Bundle sed file

SYNTAX

mapbunSed2Kml.sh <mcr\_path> <sedFile> <kmlFile>

run\_sed2kml.sh <mcr\_path>

sed2kml

DESCRIPTION

This package contains the API’s necessary to generate a Google Earth KML readable file from the Mapbundle SED file.

mapbunSed2Kml.sh is a script that allows the user to specify the sed file input fullpath filename and kml file output fullpath filename. The <mcr\_path> is the location of the MATLAB Compiler Runtime, e.g. /usr/local/MATLAB/MATLAB\_Compiler\_Runtime/v83

run\_sed2kml.sh is the linux executable script file generated by the MATLAB compiler to launch the sed2kml stand-alone executable. For automation, this file will be called by the mapbunSed2Kml.sh script. The <mcr\_path> is the location of the MATLAB runtime libraries, e.g. /usr/local/MATLAB/MATLAB\_Compiler\_Runtime/v83

sed2kml is the linux executable file generated by the MATLAB compiler to execute the sed2kml() program. In the current working directory, this program looks for a file named, sed.csv and it writes results into a file named G\_Earth\_plot.kml

File Formats:

<sedFile> contains Satellite Ephemeris Data and must not have the md5 checksum

<kmlFile> output file of contours and coordinate points of map bundle

JPEG format currently

NOTES

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/MapTesting/src/mapbun/mapbunSed2Kml.sh

//Arclight/ArcLight/AcceptanceTest/Automation/MapTesting/src/misc/run\_sed2kml.sh

//Arclight/ArcLight/Engineering/deployment/MapFiles/MapTools/KmlConversion/sed2kml.m,

#### Results Database Interface

##### Results Database Interface Manual Page

This user manual page describes the Results Database API (ReDa).

SYNOPSIS

REDA - Results Database API

SYNTAX

from RedaLibary IMPORT \*

class redaLib:

def \_\_init\_\_(self, deleteDbLog):

def addGroup(self, testStation, groupName) :

def updateGroupConfig(self, grpid, fields) :

def updateGroupResult(self, grpid, fields) :

def addVersionInfo(self, grpid, name, type, vers) :

def editReleaseCycle(self, grpid, yesNo) :

def addLogData(self, grpid, file, fields) :

def deleteLogData(self, grpid, fileFormat) :

def addSmConsoleLogData(self, grpid, file, mac, fields) :

def deleteSmConsoleLogData(self, grpid) :

def addTest(self, grpid, testName) :

def updateTestConfig(self, grpid, tstid, fields) :

def updateTestResult(self, grpid, tstid, fields) :

def getTestSpecPassFail(self, grpId) :

def getGroupNames(self) :

def getTestNames(self) :

def addGroupName(self, gName) :

def getResourceStatus

def getResourceList(self) :

def updateResource(self, type, name, fields) :

def getConsoleNames(self, name) :

def getConsolePorts(self, name) :

def addTableData(self, grpid, tstid, table, fields) :

def addIxOs(self, grpid, tstid, fields) :

def updateIxOs(self, ixosid, fields) :

def addIxOsTimeSeries(self, ixosid, fields, closeConnection) :

def addIxLoad(self, grpid, tstid, fields) :

def updateIxLoad(self, ixloadid, fields) :

def addIxLoadTimeSeries(self, type ixloadid, fields) :

def getIxOsAggTput(self, grpId, testId, direction) :

DESCRIPTION

The REDA package provides methods for managing results data in the (MySQL) database.

NOTES

EXAMPLES

SEE ALSO

FILES

//Arclight/ArcLight/AcceptanceTest/Automation/NMS/RedaLibrary.py

#### MySQL Interface

MySQL is an open source database that will be used as a COTS tool for this project. The product will be installed on the AL Test Linux device according to the instructions for RPM installation specified by reference document REF-22-01 in Section 2.2 - Commercial Standards.

Once installed, the database will be configured with a login (managed by the database connection object defined in Section 4.3.2.3.1) and the tables shown in

Figure 17, which will be created using an SQL initialization script according to the definitions detailed in Section 5.10.

The ReDa GUI interface supports phpMyAdmin which can be used to initialize and maintain the MySQL database [TBR]. Initially the SQL interpreter (command line) of MySQL will be used to create, configure and display tables; some with scripts.

MySQL

Browser

GroupInfo

ReDa Items to be modified or deprecated

CPE

CpeTimeSerics

PageTime

ObjectTime

PerfCheckSiteInfo

PerfCheckResults

PerfCheckConfig

Queries

WebTimer

TestInfo

PerfCheckTestSegmentInfo

ArcLight Config/Status

mapbunConf

nmsEmsGuiConf

termAccessConf

tuningConf

mapbunDetail

dirsDetail

nmspmVMTStatus

termVMTStatus

NlgConf

Figure - Database Tables

# CSCI detailed design

This section describes each software unit of the CSCI with sufficient detail to proceed with coding.

## Apache

Apache will launch the AL Test GUI scripts according to the URL and html from the index.html starting point.

The following special configuration is necessary:

* /etc/httpd/conf /httpd.conf will contain the following items:
  + ServerRoot "/etc/httpd"
  + LoadModule cgi\_module modules/mod\_cgi.so
  + ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
  + DocumentRoot "/var/www/html"
  + AddHandler cgi-script .cgi .py .php
  + Additions to <Directory "/var/www/cgi-bin">
    - Options All
    - AllowOverride All
    - Order allow,deny
    - Allow from all
  + SetEnv DISPLAY :47 (see Xvfb)
  + SetEnv LD\_LIBRARY\_PATH /opt/google/chrome/lib
* HTTPD Service
  + service httpd start (for starting once)
  + systemctl enable httpd.service (for starting at boot)
* Allow webserver traffic
  + firewall-cmd --zone=public --add-port=80/tcp --permanent
  + firewall-cmd --reload
  + Alternatively (if iptables instead of firewalld):
    - iptables -I INPUT 4 -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
    - iptables-save (to make permanent, i.e. after reboot)
* FIREFOX browser support
  + chgrp apache /usr/share/httpd
  + chmod g+w /usr/share/httpd

## AL Test GUI

The functions in this section implement the API specified in Section 4.3.2.2.2 AL Test GUI Scripts Interface and handle the html and processing as described below.

### Main

This page is controlled by the index.html script.

#### index.html Pseudo-Code

This is the pseudo code for the ArcLight Test GUI Homepage that renders an html page shown in Figure 9 - Homepage.

<!DOCTYPE html>

<html>

<head>

<title>Home-AL Test Automation</title>

</head>

<body>

<font size="6"> ArcLight Test Automation Homepage </font>

<font size="4">

<br><br>

<a href="/cgi-bin/altgTestLauncher.py"> Test Procedures Launch Menu </a>

<br><br>

<a href="/cgi-bin/altgTestResultsMain.py"> Test Results Menu </a>

<br><br>

<a href="/cgi-bin/altgConfScreen.py"> Test Configuration Menu </a>

<br><br>

<a href="/cgi-bin/altgUtilScreen.py"> Test Utilities Menu </a>

<br><br>

<a href="/cgi-bin/reda/index.php"> ReDa Menu </a>

</font>

</body>

</html>

### Test Case Launch

This section contains the Test GUI script detailed designs for launching tests.

#### altgTest.py Pseudo-Code

This is the pseudo code for the ArcLight Test GUI Launch that renders an html page shown in Figure 10 - Test Launch Screen.

<!DOCTYPE html>

<html>

<head>

<title>Home-AL Test Automation</title>

</head>

<body>

<font size="6"> ArcLight Test Automation Homepage </font>

<font size="4">

<br><br>

<a href="screen\_test.html"> Test Procedures Launch Menu </a>

<br><br>

<a href="screen\_status.html"> Test Results Menu </a>

<br><br>

<a href="screen\_config.html"> Test Configuration Menu </a>

</font>

</body>

</html>

#### altgTestStart.py Pseudo-Code

The altgTestStart.php script accepts the test indicator and invokes the altcAutoMgr script providing the selected parameters.

### Results

This section contains the Test GUI script detailed designs for reviewing results.

The results should provide the info found in: //Arclight/ArcLight/SystemTest/TestReport/SED\_SSCF/Bundles\_v102.42\_v106.46\_v107.52\_v108.9/Datasheet\_v102.42\_v106.46\_v107.52\_v108.9.xlsx

#### altgStatus.py Pseudo-Code

This is the pseudo code for the ArcLight Test GUI Results that renders an html page shown in Figure 11 - Test Results Screen.

[TBR - This is the entry-point where the ReDa web interface can be activated.]

<!DOCTYPE html>

<html>

<head>

<title>Results-AL Test Automation</title>

</head>

<body>

<br><br>

<form action="altgStatusDetail.php">

<fieldset>

<legend>ArcLight Test Automation Results</legend>

<br>

<table>

<head>

<style>

table, th, td {

border: 1px solid black;

border-collapse: collapse;

}

</style>

</head>

<table style="width:100%">

<tr>

<th>Id</th>

<th>Group Id</th>

<th>Timestamp</th>

<th>Status</th>

<th>Description</th>

<th>Details</th>

</tr>

<tr>

<td align="right">133</td>

<td align="right">200</td>

<td align="center">2015-04-06 13:10:15</td>

<td align="center">Passed</td>

<td align="center">Map Trickle</td>

<td align="center"><input type="button" onclick="alert('Loading details page for this iteration')" value="Details"></td>

</tr>

<tr>

<td align="right">134</td>

<td align="right">200</td>

<td align="center">2015-04-06 15:55:15</td>

<td align="center">In-Progress</td>

<td align="center">Map Term Download</td>

<td align="center"><input type="button" onclick="alert('Loading details page for this iteration')" value="Details"></td>

</tr>

</table>

</fieldset>

</form>

<br><br>

<form action="screen\_status.html">

<input type="submit" value="Refresh">

</fieldset>

</form>

</body>

</html>

#### altgStatusDetail.py Pseudo-Code

The altgStatusDetail.php script accepts the test indicator and invokes the altcAutoMgr script providing the selected parameters.

### Configuration

This section contains the Test GUI script detailed designs for test configuration.

The config page reads the database (or default items) based on Group Id and populates a config screen. Once the config screen is opened, the save/submit buttons for each section, e.g. Bundle config, NMS/EMS config, and Term config, should not “disturb” the other sections. This is most easily done by having a print routine that has a parameter list containing all the fields and then the procedure can be invoked from database get calls or by using the GET/POST parameters.

The configuration tuning parameters are used to tune software algorithms that are “hardcoded” or “formulamatic” to these parameters, i.e. a runtime include file. The goal is that the developer uses these parameters in the code, and modifies these parameters infrequently.

#### altgConfig.py Pseudo-Code

This is the pseudo code for the ArcLight Test GUI Configuration that renders an html page shown in Figure 14 - Configuration Screen

<!DOCTYPE html>

<html>

<head>

<title>Config-AL Test Automation</title>

</head>

<body>

<br><br>

<form action="screen\_config\_mapbundle.html"

enctype="multipart/form-data" method="post">

<fieldset>

<legend>ArcLight Test Automation Map Bundles Configuration</legend>

<p>

Map Bundle File:

<input type="file" name="datafile" size="400">

</p>

<table>

<tr>

<td align="right">FL Profile:</td>

<td align="left"><input type="text" name="mapbprofile" value="SL-ACSM">

</td>

</tr>

<tr>

<td align="right">Install Script:</td>

<td align="left"><input type="text" name="mapbscript" value="scriptReboot-1.0.0.3.sh">

</td>

</tr>

<tr>

<td align="right">Satellite Id(new):</td>

<td align="left"><input type="text" name="mapbsatnew" value="21">

</td>

</tr>

</table>

<div>

<input type="submit" value="Save">

</div>

</fieldset>

</form>

<br>

<form action="screen\_config\_nmsaccess.html">

<fieldset>

<legend>ArcLight Test Automation NMS/EMS GUI Access Configuration</legend>

<table>

<tr>

<td align="right">IP:</td>

<td align="left"><input type="text" name="nmsip" value="192.168.111.254">

</td>

</tr>

<tr>

<td align="right">Port:</td>

<td align="left"><input type="text" name="nmsport" value="9090">

</td>

</tr>

<tr>

<td align="right">Username:</td>

<td align="left"><input type="text" name="nmsuname" value="altest">

</td>

</tr>

<tr>

<td align="right">Password:</td>

<td align="left"><input type="password" name="nmspw" value="viasat123">

</td>

</tr>

</table>

<br><br>

<input type="submit" value="Save">

</fieldset>

</form>

<br>

<form action="screen\_config\_termaccess.html">

<fieldset>

<legend>ArcLight Test Automation Terminal Access Configuration</legend>

<table>

<tr>

<td align="right">IP:</td>

<td align="left"><input type="text" name="termip" value="192.168.111.254">

</td>

</tr>

<tr>

<td align="right">Port:</td>

<td align="left"><input type="text" name="termport" value="9090">

</td>

</tr>

<tr>

<td align="right">Username:</td>

<td align="left"><input type="text" name="termuname" value="altest">

</td>

</tr>

<tr>

<td align="right">Password:</td>

<td align="left"><input type="password" name="termpw" value="viasat123">

</td>

</tr>

<tr>

<td align="right">Terminal Name:</td>

<td align="left"><input type="text" name="termname" value="Terminal 04 - 82.1">

</td>

</tr>

<tr>

<td align="right">Terminal CLI:</td>

<td align="left"><input type="radio" name="termcmd" value="serial" checked>Console</td>

<td align="left"><input type="radio" name="termcmd" value="ssh" checked>SSH</td>

</tr>

<tr>

<td align="right">Console Port:</td>

<td align="left"><input type="text" name="serial" value="terminal\_4">

</td>

</tr>

</table>

<br><br>

<input type="submit" value="Save">

</fieldset>

</form>

<br><br>

<a href="screen\_config\_tuning.html"> Test Automation Tuning Parameters </a>

<br><br>

<form action="screen\_config\_clear.html">

<input type="submit" value="Reset">

</fieldset>

</form>

</body>

</html>

#### altgConfigMap.py

Save the GUI Map Configuration parameters in the ALDB Configuration Package using the API’s provided

Print out all the GET/POST input parameters from the config page.

#### altgConfigNms.py

Save the GUI NMS/EMS Configuration parameters in the ALDB Configuration Package using the API’s provided

Print out all the GET/POST input parameters from the config page.

#### altgConfigTerm.py

Save the GUI Terminal Configuration parameters in the ALDB Configuration Package using the API’s provided

Print out all the GET/POST input parameters from the config page.

#### altgConfigTuning.py Pseudo-Code

HMTL code is below. Use the individual tuning parameters as follows:

* GroupID
  + (READ-ONLY)
* precedenceDuration and precedenceDwell
  + Modify the
* profileEnabled =
* forwardLinkName= (<base>\_<groupId>\_<mapVersion>)
* defaultBitrate= (kbps)
* downloadBitratePercentage=
* hubName=

<!DOCTYPE html>

<html>

<head>

<title>Tuning-AL Test Automation</title>

</head>

<body>

<br><br>

<form action="screen\_config\_mapbundle.html"

enctype="multipart/form-data" method="post">

<fieldset>

<legend>ArcLight Test Automation Tuning Parameters</legend>

<table>

<tr>

<td align="right">Precedence Duration (sec):</td>

<td align="left"><input type="text" name="tunepresdur" value="900">

</td>

</tr>

<tr>

<td align="right">Precedence Dwell (sec):</td>

<td align="left"><input type="text" name="tunepresdwell" value="60">

</td>

</tr>

</table>

<br><br>

<div>

<input type="submit" value="Save">

</div>

</fieldset>

</form>

</body>

</html>

### Utilities

This section contains the Test GUI script detailed designs for utilities.

The utilities page provides sections for configuring and executing utilities.

Currently only the map test report utility is defined, and is used to glean results from test cases for the map bundle QA test report.

* The combined upload/trickle/flock/precedence report is used to create inputs for the Excel Test Report for QA
* The compare report is used to report differences in old and new versions of a map/variant.
* The signed report is used to verify signed maps against unsigned maps by verifying MD5 checksums are the same between bundles for Upload and Trickle.

#### Pseudo-Code

#!/bin/bash

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

### utilsMapResults.sh

### This script generates a report of map test results

### for specified groups

###

### Input parameters:

### -o old (unsigned) group list

### -n new (signed) group list

### -c (| --compare) compare sed/sscf differences

### -f (| --fllock) report fllock

### -p (| --precedence) report precedence

### -s (| --signed) compare unsigned/signed checksum differences

### -t (| --trickle) report trickle

### -u (| --upload) report upload

###

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

compar=0

fllock=0

prec=0

trickle=0

upload=0

while [ ! $# -eq 0 ]

do

case "$1" in

-c | --compare)

compar=1

;;

-f | --fllock)

fllock=1

;;

-n)

group\_new=(`echo $2 | sed '{s/,/ /g;s/\[//;s/\]//}'`)

shift

;;

-o)

group\_old=(`echo $2 | sed '{s/,/ /g;s/\[//;s/\]//}'`)

shift

;;

-p | --precedence)

prec=1

;;

-s | --signed)

signver=1

;;

-t | --trickle)

trickle=1

;;

-u | --upload)

upload=1

;;

esac

shift

done

## AL Database

### ALDBCONN

The database connection information is accessible through ALDBCONN and used by the ReDa and/or AL DB database management layers. If the data is needed in multiple script languages, then a generator script can read in the single input file and generate files for multiple script languages.

#### Pseudo-Code

from aldbconnConfig IMPORT \*

### Data dictionary optimized for MySQL with tags used by connection API

def aldbConnParmsGet ():

return { 'user':dbuser, 'password':dbpassw, 'host':hostname, 'port':portnum 'database':dbname}

### ALDBCONF

ALDBCONF functions shuttle data between python and MySQL. The pseudo-code below shows and example of inserting/querying a record to/from MySQL.

#### Pseudo-Code

from \_\_future\_\_ import print\_function

from datetime import date, datetime, timedelta

import mysql.connector

class aldbConf(object):

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

### init()

### <add description here>

###

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

def \_\_init\_\_(self, groupId):

self.groupId=groupId

self.connParms=aldbConnParmsGet()

cmd = mkdir -p resultsBaseDir/groupId

result = os.system(cmd)

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

### aldbConfMapbunSet ()

### write MAPBUN\_CONF to database

###

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

def aldbConfMapbunSet(self, filename, profile, script, satid):

cnx = mysql.connector.connect(host=self.connParms[‘dbhostname’], port=self.connParms[‘dbportnum’], database=self.connParms[‘dbname’], user=self.connParms[‘dbusername’], password= self.connParms[‘dbpassw’])

cursor = cnx.cursor()

### NOTE: Table constraint “ADD UNIQUE groupId”

add\_mapbun\_conf =

("INSERT INTO MAPBUN\_CONF "

"(groupId, filename, profile, script, satId)"

"VALUES (%(groupId)s, %(filename)s, %(profile)s, %(script)s, %(satId)s)"

“ON DUPLICATE KEY UPDATE VALUES (%(filename)s, %(profile)s, %(script)s, %(satId)s) )

### Insert mapbun\_conf information

data\_mapbun\_conf = {

‘groupId’: self.groupId,

'filename': filename,

'profile': profile,

'script': script,

'satId': satid,

}

cursor.execute(add\_mapbun\_conf, data\_mapbun\_conf)

# Make sure data is committed to the database

cnx.commit()

cursor.close()

cnx.close()

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

### aldbConfMapbunGet ()

### read MAPBUN\_CONF from database

###

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

def aldbConfMapbunGet(self)

cnx = mysql.connector.connect(host=self.connParms[‘dbhostname’], port=self.connParms[‘dbportnum’], database=self.connParms[‘dbname’], user=self.connParms[‘dbusername’], password= self.connParms[‘dbpassw’])

cursor = cnx.cursor()

### NOTE: Table constraint “ADD UNIQUE groupId”

query\_mapbun\_conf =

("SELECT filename, profile, script, satid FROM MAPBUN\_CONF"

"WHERE groupId=%s")

cursor.execute(query\_mapbun\_conf, self.groupId)

for (filename, profile, script, satid) in cursor:

result = (‘filename’:filename, ‘profile’:profile, ‘script’:script, ‘satId’:satid)

cnx.commit()

cursor.close()

cnx.close()

return result

## AL Test Cases

The AL Test Cases entity is comprised of the AL Automation Manager and the Map Bundles entities.

### AL Automation Manager

The AL Automation manager is invoked from the AL Test GUI to launch test case(s).

#### Pseudo Code

class alam(object):

def \_\_init\_\_(self):

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

### alamTestStart ()

### <add description here>

###

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

def alamTestStart(self, testType, testId, groupId):

map = mapbun(groupId)

map.mapbunConf(testid)

if (testId = ALAMTESTTYPE[‘trickle’] OR testId = ALAMTESTTYPE[‘all’]):

map.mapbunTrickle(testId)

if (testId = ALAMTESTTYPE[‘upload] OR testId = ALAMTESTTYPE[‘all’]):

map.mapbunUpload(testId)

if (testId = ALAMTESTTYPE[‘fllock] OR testId = ALAMTESTTYPE[‘all’]):

map.mapbunFlLock(testId)

if (testId = ALAMTESTTYPE[‘precedence] OR testId = ALAMTESTTYPE[‘all’]):

map.mapbunPrecedence(testId)

return

### Map Bundles

This class automates the Map Bundles Test Procedure (REF-23-02)

Private member data for internal-use only across all mapbun methods: groupId, conf, nmsConf, termConf, mapConf, mapDirConf, nms, ems, term, zipFile, isSigned, version, antennaType, antennaLoFreq, emsTxFreq, termRxFreq, termLbandFreq, chipRate, latitude, longitude

TBR Items:

* May need to monitor the terminal with tail –f /test/syslogmsg
* Need an initial config with backup/restore for an operational checkpoint.

#### \_\_init\_\_

mapbun class initialization of the sets the following member data for use within the class:

* groupId
* conf (aldbConf for the groupId)
* nmsConf (aldbNmsConfGui for the groupId)
* termConf (aldbTermConfGui for the groupId)
* mapConf (aldbConfMapbunDetailed for the groupId)
* mapDirConf (aldbConfDirs for the groupId)
* nms (algNms class instance)
* ems (algEms class instance)
* term (algTerm class instance)

#### Pseudo Code

### IMPORT the classes used by the mapbun object

from aldbConn IMPORT \*

from aldbConnConfig IMPORT \*

from aldbConf IMPORT \*

from viasatSelenium IMPORT \*

from alGui IMPORT \*

from TermSerialCom IMPORT \*

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

### \_\_init\_\_ ()

### <add description here>

###

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

def \_\_init\_\_(self, groupId):

self.grpId = groupId

self.conf = aldbConf(self.grpId)

self.nmsConf = self.conf.aldbNmsConfGuiGet()

self.termConf = self.conf.aldbTermConfGuiGet()

self.termMapverConf = self.conf.aldbConfMapbunDetailedGet()

self.mapConf = self.conf.aldbConfMapbunGet()

self.mapDirConf = self.conf.aldbConfDirsGet()

self.nms = algNms(“http://”+ str(self.nmsConf[‘ip’]) + “:” + str(self.nmsConf[‘port’]), self.nmsConf[‘usern’], self.nmsConf[‘passw’],

self.mapDirConf[‘resultsDir’])

self.ems = algEms(“http://”+ str(self.nmsConf[‘ip’]) + “:” + str(self.nmsConf[‘port’]), self.nmsConf[‘usern’], self.nmsConf[‘passw’],

self.mapDirConf[‘resultsDir’])

self.term = algTerm((“http://”+ str(self.termConf[‘ip’]) + “:” + str(self.termConf[‘port’]),,self.termConf[‘usern’], self.termConf[‘passw’], self.mapDirConf[‘resultsDir’])

#### mapbunConf

The mapbunConf member function prepares the configuration of the mapbun instance by:

* Copying the map bundle over to the destination associated with the Group Id
* Run a script that
  + prepares the bundle for the NMS and Terminal uploads by
    - unziping the bundle; removes certificate and leaves zip file used for NMS
    - unzipping the bundle; gives SED, SSCF, GDRM and RLC files for Term
  + prepares each file by copying and strips out checksum
    - Copy sed.csv.agt to sed.csv, sscf.csv.agt to sscf.csv, rlc.txt.agt to rlc.txt and gdrm.txt.agt to gdrm.txt
    - Remove the checksum line from each file and save it as the included checksum
    - Run md5 and update database with md5 checksum and compare to the included checksum
  + Generates a kml Google Earth map file using sed2kml.exe
* Run a script file utilizing sed/awk on the sed.csv map file to extract antenna type and based on the satellite the frequency (for computing FL and Term RF frequencies) and chip rate
  + sed.csv
    - Record Type 1 is Name; match name to get Id
    - Record Type 3 is Freq
    - Record Type 9 is Chip Rate
    - Example:
      * #
      * # ASTRA-2E\_223(26.55/Y/A) Satellite #21
      * #
      * 21,0,3,3,0
      * 21,1,1,ASTRA-2E\_223(26.55/Y/A),2820,0,1
      * 21,3,H,144750,116750,H,144750,116750
      * 21,9,26550,0,2,A011,0001,A011,0002,1,0,2,200001,17e097,1
* Run the script that generates the map coordinates files
  + Run matlab on linux in nodesktop mode; can the license be used?
  + So that the Latitude/Longitude can be determined using a coordinate pair with the fewest intersections from the optimized file
  + And the files will be available for the precedence test
  + //Arclight/ArcLight/AcceptanceTest/Automation/Scripts/Maps/find\_coords/find\_coords.m
* This method sets the following private member data items:
  + zipFile
  + isSigned
  + version
  + antennaType
  + antennaLoFreq
  + emsTxFreq
  + termRxFreq
  + termLbandFreq
  + chipRate
  + latitude
  + longitude
* **NOTE: If the groupId has already been processed, then mapbunConf will set the private member data items (but not recreate the already unzipped files)**

##### Pseudo Code

Pseudo code for mapbunConf, Prep Files, Sed Parameter script, map bundle coordinates, Map Parameter extract.

###### mapbunConf

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

### mapbunConf()

### <add description here>

###

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

def mapbunConf(self):

if Already Extracted

self.mapDetailConf=aldbConfMapbunDetailedGet(self)

else

### Move file to config Dir

### Extract the files

### Strip out the checksum for the files

### Extract parameters from sed file

cmd = “mapbunPrepFiles.sh ” + str(self.mapConf[‘filename’]) + “ “ + self.mapDirConf[‘configDir’] + str(self.mapConf[‘bundleVersionNum’] + str(self.mapConf[‘satName’])

result = os.system(cmd)

self.conf.aldbConfMapbunDetailedSet(result)

endif

self.antennaLoFreq = antennaLoFreqGet(result[‘antennaType’])

self.termLbandFreq = result[‘flRxFreq’]- self.antennaLoFreq

satlistFile = “satlist\_v” + str(self.mapConf[‘bundleVersionNum’] + “.txt”)

precbyidFile = “coords\_precid\_v” + str(self.mapConf[‘bundleVersionNum’] + “.csv”)

precbynameFile = “coords\_precname\_v” + str(self.mapConf[‘bundleVersionNum’] + “.csv”)

coordsFile = “coords\_v” + str(self.mapConf[‘bundleVersionNum’] + “.txt”)

mapFile = “map\_v” + str(self.mapConf[‘bundleVersionNum’] + “.pdf”)

### Generate Coordinate Files

coordfile = open(self.mapDirConf['configDir'] + satlistFile)

coordfile.write(“{}\n”, self.mapConf[‘satName’])

cmd = “run\_findAllCoords.sh ” + self.tuningConf['matlabLibPath'] + “ “ + self.mapDirConf['configDir'] + sedFile + ” “ + self.mapDirConf['configDir'] + sscfFile + “ “ + self.mapDirConf['configDir'] + satlistFile + “ “ + self.mapConf[‘bundleVersionNum’] + “ “ + self.mapDirConf['resultDir'] + precbyidFile + “ “ + self.mapDirConf['resultDir'] + precbynameFile +

“ “ + self.mapDirConf['configDir'] + coordsFile + “ “ + self.mapDirConf['resultDir'] + mapFile”

result = os.system(cmd)

### Generate kml file for this bundle

### Extract lat/long with least number of intersections from optimized coordinate file

fp = open(precbyidFile)

result = fp.read()

self.latitude = result[‘latitude’]

self.longitude = result[‘longitude’]

###### mapbunPrepFiles.sh

#!/bin/sh

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

### mapbunPrepFiles.sh

### <add description here>

###

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

filedir=$1

filename=$2

bunver=$3

satName=$4

myhome=`pwd`

cd $filedir

# Unzip the files

unzip $filename

unzip \*$bunver\*.zip

#move the bundle file here

mv $filename .

#Extract the version numbers

for $fn in $MapFileNameList

mapVerNumArray[$count]=`sed -rn 's/.\*ersion[^0-9]+([0-9\.]+)/\1/p' $fn`

# Make the stripped files and save md5

if [ -e sed.csv.agt ] && [ ! -e sed.csv]; then

md5sed=`tail -1 sed.csv.agt | sed <TBD>`

sed '$ d' sed.csv.agt > sed.csv

fi

if [ -e sscf.csv.agt ] && [ ! -e sscf.csv]; then

md5sscf=`tail -1 sscf.csv.agt | sed <TBD>`

sed '$ d' sscf.csv.agt > sscf.csv

fi

if [ -e rlc.txt.agt ] && [ ! -e rlc.txt]; then

md5rlc=`tail -1 rlc.txt.agt | sed <TBD>`

sed '$ d' rlc.txt.agt > rlc.txt

fi

if [ -e gdrm.txt.agt ] && [ ! -e gdrm.txt]; then

md5gdrm=`tail -1 rlc.txt.agt | sed <TBD>`

sed '$ d' gdrm.txt.agt > gdrm.txt

fi

#Get the Antenna Type

# Per Email - based on Map Version

# From: Khuc, An

# Sent: Thursday, August 06, 2015 12:11 PM

# Subject: RE: Antenna type for sed in 102.43

# 102.x, 107.x, 108.x, 3.x are for VR-12 Ku

# 6.x for Rantec

# 1.x for KVH V7

# 5.x for KVH V3

# 7.x for KVH V11

# 106.x for both TeCOM and VR-12 Ku

#Get the Freq & Chip Rate Info; divide freq’s by 10

#Create and print the dictionary

cd $myhome

###### mapbunSed2Kml.sh

#!/bin/sh

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

### mapbunCoordinates

### This script invokes the MATLAB Standalone

### sed2kml program for generating the

### Google Earth KML file. Since the sed2kml is

### from the engineering baseline and relies on

### hardcoded filenames, this script provides

### a wrapper with additional flexibility

###

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

#### mapbunTrickle

The mapbunTrickle member function completely executes the trickle test by:

* Using the map associated with the class
* Uploading to NMS
* Downloading to the Terminal, use reset script
* Verifying that the correct map version is running after terminal relocks onto FL
* Handling normal and error case results reporting

##### Pseudo Code

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

### mapbunTrickle()

### <add description here>

###

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

def mapbunTrickle(self, testId):

result = FAILED

state = START

self.nms.algNmsLogin()

self.term.algTermLogin()

self.nms.algNmsDnldMapUpload(self.zipFile, self.isSigned, self.antennaType)

state = UPLOAD

self.nms.algNmsDnldMapProfile(enabled, defBitRate, fl, self.mapConf[‘script’], dnldBitRatePercent, vmtList)

state = DOWNLOAD

### Wait for the trickle download to complete

while !timeout:

if self.nms.algNmsDnldMapStatus[‘done’]:

state = SENT

break

### Wait for the terminal to reset and re-acquire

### [TBR - monitor the terminal with tail –f /test/syslogmsg]

if state == SENT:

while !timeout:

termFLStatus = self.term.algTermStatusFLLock()

if termFLStatus[‘flState’] == ‘Locked’:

state = LOCKED

break

### Verify the new map bundle is operative

if state == LOCKED:

termVerStatus = self.term.algTermStatusVersion()

if termVerStatus [‘sedVersion’] == termMapverConf [‘sedVersion’] AND

termVerStatus [‘sscfVersion’] == termMapverConf [‘sscfVersion’] AND

termVerStatus [‘rlcVersion’] == termMapverConf [‘rlcVersion’] AND

termVerStatus [‘gdrmVersion’] == termMapverConf [‘gdrmVersion’]:

result = PASSED

else

result = FAILED

### Take a screenshot of the versions and place into results

self.term.saveScreenshot((self.mapDirConf[‘resultDir’] + “/mapbunTrickleTermVers.png”))

self.nms.algNmsLogout()

self.term.algTermLogout()

### Post results for the group/test

### [TBR]

### return status

return(result, state)

#### mapbunUpload

The mapbunUpload member function completely executes the Terminal Upload test by:

* Uploading the maps (.csv, .txt) associated with the class through the Terminal GUI connected to local LAN port
  + Configuration Tab-> Satellite (Left Pane)->SED [SSCF | RLC | GDRM] file->Choose File
  + Choose Upload button
* Verifying upload with “Read File” name versions

##### Pseudo Code

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

###

### mapbunUpload()

### <add description here>

###

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

result = FAILED

state = START

### Login to the Terminal

self.term.algTermLogin()

### Upload the map files to the terminal

result = mapbunTermFilesGet()

algTermMapUpload(result[‘sedfile’], result[‘sscffile’], result[‘rlcfile’], result[‘gdrmfile’])

### Verify the new map bundle versions displayed

if state == LOCKED:

termVerConf = self.term.algTermConfVersion()

if termVerConf [‘sedVersion’] == self.termMapverConf [‘sedVersion’] AND

termVerConf [‘sscfVersion’] == self.termMapverConf [‘sscfVersion’] AND

termVerConf [‘rlcVersion’] == self.termMapverConf [‘rlcVersion’] AND

termVerConf [‘gdrmVersion’] == self.termMapverConf [‘gdrmVersion’]:

result = PASSED

else

result = FAILED

### Take a screenshot of the versions and place into results

term.saveScreenshot((self.mapDirConf[‘resultDir’] + “/mapbunUploadVers.png”))

### Logout of the terminal

term.algTermLogout()

### Post results for the group/test

### [TBR]

#### mapbunFLLock

The mapbunFLLock member function completely executes the Forward Link Lock test by:

* Reconfiguring the EMS to use the new Forward Link frequency and chip rate
* Reconfiguring the Terminal to a Latitude/Longitude within the new beam
* Reconfiguring the Terminal to the antenna specified in sed.csv
* Rebooting the terminal
* Verifying the Terminal locks to the FL for the correct beam
* (BR-Cband Spectral inversion check FWL test procedure)

Note, use //Arclight/ArcLight/AcceptanceTest/Automation/Scripts/Maps/SED\_SSCF\_bundtest.py as a working example for using the TermCmd object.

##### Pseudo Code

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

### mapbunFLLock()

### <add description here>

###

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

def mapbunFLLock (self, testId):

result = FAILED

state = START

self.ems.algEmsLogin()

self.ems.algEmsAcsmFlUpdate(self.mapConf[‘profile’], hub, self.emsTxFreq, self.termRxFreq, self.chipRate)

state = FLTX

### Set the terminal lat/long and reboot to force the FL search

### TBR - setup the console/terminal class

termCmd.login

termCmd.setLatLong(self.latitude, self.longitude)

termCmd.setAntenna(self.antennaType)

termCmd.reboot()

state = REBOOT

### Wait for the terminal to reset and re-acquire

### [TBR - monitor the terminal with tail –f /test/syslogmsg]

if state == REBOOT:

self.term.algTermLogin()

while !timeout:

termFLStatus = self.term.algTermStatusFLLock()

if termFLStatus[‘flState’] == ‘Locked’:

state = LOCKED

break

### Verify the correct satellite/beam is locked

if state == LOCKED:

if termFlStatus [‘lbandFreq’] == self.termLbandFreq:

result = PASSED

else

result = FAILED

### Take a screenshot of the versions and place into results

self.term.saveScreenshot((self.mapDirConf[‘resultDir’] + “/mapbunTrickleTermVers.png”))

self.nms.algNmsLogout()

self.term.algTermLogout()

### Post results for the group/test

### [TBR]

### return status

return(result, state)

#### mapbunPrecedence

The mapbunPrecedence member function completely executes the Precedence test by:

* Changing the lat/long coordinates of the terminal based coordinates file created by mapConf
* Turning on sat search
* Removing the existing syslogMsgs file
* Rebooting the Terminal
* Waiting until Terminal is comes back up
* Creating a log file (or results database entries) of the satellites that the terminal tries to lock onto for each zone (along with Zone coordinates) by parsing the syslog Msgs containing console outputs (tempLog).
* Placing Log file is results directory for later examination for correctness of satellite locking order.
* A script will parse the log and create a “log” CSV file.
* A verify member function will use that “log” csv file can be used in conjunction with a precedence file (created at the time of the coordinates file) to verify precedence

Note, use //Arclight/ArcLight/AcceptanceTest/Automation/Scripts/Maps/SED\_SSCF\_bundtest.py as a working example for this script.

##### Pseudo Code

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

### mapbunPrecedence()

### <add description here>

###

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

def mapbunPrecedence (self, testId):

### Set the terminal lat/long and reboot to force the FL search

### TBR - setup the console/terminal class

termCmd.login()

zoneid=0

while (result = mapbunGetNextCoord()) != NULL:

zoneid++

termCmd.setLatLong(result[‘latitude’], result[‘longitude’])

termCmd.setAntenna(self.antennaType)

termCmd.rmFile ('/test/syslogMsgs')

termCmd.reboot()

termCmd.login()

time.sleep(600)

yield termCmd.grepLog('Set Antenna') #grep the syslogMsgs file for this arg

self.log.close()

### Parse log for lines where sat locks for this Zone and

### append to results file

f = open(self.mapDirConf[‘resultDir’] + “/mapbunPrecedence.log”, "a")

f.write("\nZone "+str(zoneid)+": lat "+ result[‘latitude’] +", long "+ result[‘longitude’])

f.write("--------------------------------------------------\n")

for line in open(self.capFilename, "r"):

if "Set Antenna" in line:

if "grep" in line:

pass

else:

f.write(line)

### Add to results database

f.close()

###### mapbunPrecLogExtract.sh

#!/bin/sh

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

### mapbunPrecLogExtract

### This script extracts the

### LAT/LONG and beam precedence info from the

### Terminal log in order to verify the

### Terminal follows precedence

###

### Inputs: input file name, output file name

###

### Note: File names are complete paths

###

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

#!/bin/bash

infn=$1

outfn=$2

cat $infn | sed -e '{/Zone /{s/.\*:/Zone\_/;s/lat//;s/long//;s/ //g;N;s/\n//;s/--.\*/,/}}' -e t -e '{/Antenna State/{s/.\*sat:1,//;s/ .\*/,/}}' | awk '/Zone\_$/ {print;next;} {printf("%s",$0);}' | sed -e 's/,Zone\_/\n/g' | sed -e 's/Zone\_//' | sed -e '$s/,$/\n/' | sed -e 's/,/, /g' > $outfn

### NLG

This class automates the NLG Terminal Status gathering as outlined in test plan (REF-23-06)

The NLG “poller,” shown in Figure 18- NLG Poller Architecture, operates as follows:

* Turn On/Off from ALTG
  + On
    - Launched to background from ALAM
    - Save Linux Task Id (in NLG db)
  + Off
    - Signal SIGUSR1 to Linux Task Id from NLG db lookup by groupId
* Config
  + Mode
    - Terminal
    - Both Terminal & NMSPM
  + Tuning Parameters static in alDefaults
    - Terminal Poll Rate
    - NMSPM Poll Rate
* Inputs
  + Poll Timeout
    - Terminal
    - NMSPM
  + Signal
    - SIGUSR1
      * Save files and exit
* Outputs
  + ALDB API
    - Save Terminal Status; key=group, time
    - Save NMSPM Status; key=group, time
  + Results
    - File Save of status data during test duration as CSV
    - Save Charts as PNG; select period based on test duration
    - Syslogs
      * Post Processed as events in CSV file
      * Compress Syslogs

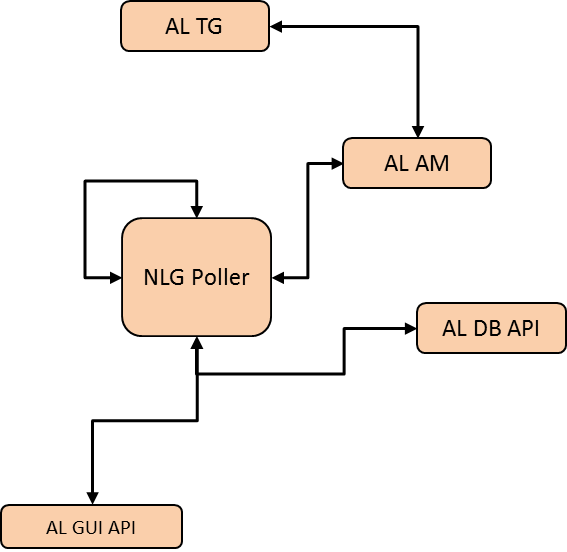


Figure - NLG Poller Architecture

### NLG Main Pseudo-Code

# Initialize class; connect to configuration

def \_\_init\_\_(self,groupId, termonly=False):

# Setup the NMSPM page for fast polling

def nlgNmspmOpen(self):

# Setup the Terminal page for fast polling

def nlgTermOpen(self):

# close the NMSPM page and generate results

def nlgNmspmClose(self):

# close the Terminal page and generate results

def nlgTermClose(self):

# Start the terminal and if necessary the nmspm

def start(self):

# Start NMSPM Data collection data

def nlgNmspmStart(self):

self.nlgNmspmOpen()

self.timerNmspmState = True

# Schedule the Timer Thread

self.timerNmspm = Timer(defNlgTimeoutNmspm,self.nlgNmspmTimerThread)

# Start Terminal Data collection data

def nlgTermStart(self):

self.nlgTermOpen()

self.timerTermState = True

# Schedule the Timer Thread

self.timerTerm = Timer(defNlgTimeoutTerm,self.nlgTermTimerThread)

# Stop the nmspm & terminal

def stop(self):

# Stop NMSPM Data collection

def nlgNmspmStop(self):

# Stop Terminal Data collection

def nlgTermStop(self):

# Signal Handler

# Upon receipt of SIGUSR1; stop the running poll processor

def nlgSigHandler(self, signum, stack):

# Poll the NMSPM Terminal (VMT) Data and add entry to the database

# Assumes NMS Map window opened

def nlgNmspmPollStatus(self):

# Poll the Terminal Data and add entry to the database

# Assumes Terminal window opened

def nlgTermPollStatus(self):

# Timer Thread for collecting NSPM data

def nlgNmspmTimerThread(self):

if self.timerNmspmState :

self.timerNmspm = Timer(defNlgTimeoutNmspm,self.nlgNmspmTimerThread)

# Timer Thread for collecting Term data

def nlgTermTimerThread(self):

if self.timerTermState :

self.timerTerm = Timer(defNlgTimeoutTerm,self.nlgTermTimerThread)

# NLG VMT Logout Command

def nlgNmsVmtCommand(self, command):

# Start a backgroun running poller task for this group

def nlgStartPoll(self):

# Send signal to the running poller task for this group

def nlgStopPoll(self):

### NLG Syslog Post Processing

Upon closing NLG NMSPM and Terminal threads, the RTNMS and Terminal syslogs are scp’ed back to the results. Those syslogs are converted to .csv files for easier analysis in Excel using a post processing script called from nlgMain.py. Below are the RTNMS and Terminal Syslog pseudo code descriptions

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

### nlgPostprocRtnms.sh

###

### This script parses the Rtnms syslog file

### into an event oriented CSV file w/Header Row

### Currently, the following events are parsed:

### VMT\_STATE,CRL\_STATE,CONN\_LOGIN,CONN\_OPER,

### CONN\_SESS,PWRCTL\_UPD,ACSM\_UPD,CONN\_LOGOUT,

### CONN\_Vmt,CONN\_VMT,CONN\_RL"

### Each event is represented with a "1" in the event column,

### and any supporting info is in the VARG column(s) as

### follows:

### VARG\_1 is Terminal ID (decimal equiv of 0.0.c.d addr), or

### CRL indicator for CRL\_STATE

### VARG\_2 is State for VMT\_STATE or CRL\_STATE, otherwise

### contains message

###

### Input parameters:

### basedir to put files

### input file name (in basedir or full path)

### output file name (to basedir)

###

### ArcLight Test Automation

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

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

### nlgPostprocTerm.sh

###

### This script parses the terminal syslog file

### into an event oriented CSV file w/Header Row

### Currently, the following events are parsed:

### SAT\_SRCH, SAT\_LOCK, BRL\_LOGIN, CRL\_LOGIN,

### LOGOUT\_CMD, REBOOT\_CMD, RELOGIN\_CMD, TOD\_RESET,

### SET\_ANT, DWELL\_TIMEO, SAT\_HANDO, RLC\_STATE,

### SMS\_STATE, BB\_STATE, LOGIN\_STATE, TERM\_STATE,

### UPD\_ANTTXB, SEND\_ANTSAT, MODE\_CHG

### Each event is represented with a "1" in the event column,

### and any supporting info is in the VARG column(s) as

### follows:

### If TOD\_RESET, then VARG\_1 is TOD\_NEW and VARG\_2 is TOD\_OLD

### If SET\_ANT, then VARG\_1 is BEAM\_NAME and VARG\_2 is 0

### If \*\_STATE, then VARG\_1 is state change value and VARG\_2 is 0

### If SEND\_ANTSAT, then VARG\_1 is AcuSt and VARG\_2 is 0

### If UPD\_ANTTXB, then VARG\_1 is AcuSt and VARG\_2 is 0

###

### Input parameters:

### basedir to put files

### input file name (in basedir or full path)

### output file name (to basedir)

###

### ArcLight Test Automation

###

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

The following syslog to event mapping is used:

* Satellite Search (SAT\_SRCH)
  + daemon.debug CCD: [SSM:NO COVER:03] SAT #Avail: 1
* Satellite Lock (SAT\_LOCK)
  + daemon.info CCD: [SSM:SAT LOCK:01] SATSRCH STATE CHANGE: \*\* SAT LOCK \*\*
* (BRL\_Login)
  + daemon.info CCD: Received CRL PCM after Login response PCM Ack sent - Assume officially logged in via BRL now (t=1432912130 csec)
* (CRL\_Login)
  + daemon.info CCD: [LSM:07:01] LOGIN STATE CHANGE: \*\* Terminal Logged In (CRL) \*\*
* Logout Command (LOGOUT\_CMD)
  + CCD: \[ACP:10:03\] ACP Command Message -- LOGOUT CMD --
* Reboot Command (REBOOT\_CMD)
  + CCD: \[ACP:10:01\] ACP Command Message -- REBOOT CMD –
* Relogin Command (RELOGIN\_CMD)
  + CCD: \[ACP:10:02\] ACP Command Message -- RELOGIN CMD --
* Time of Day Reset (TOD\_RESET)
  + CCD: Time of Day Reset.
* Set Antenna (SET\_ANT)
  + CCD: \[SSM:NO COVER\] Set Antenna State sat:“
* Dwell Expired
  + daemon.debug CCD: [SSM:HandleDwellTimerExp] SEARCH: entering
  + daemon.info CCD: [SSM]:next satsearch
* Handover
* daemon.info CCD: [SSM] Satellite Handover, satDistance
* State Changes
  + daemon.info CCD: [RLC:00:01] RLC MSG STATE CHANGE: FAIL
  + daemon.info CCD: [BBM:00:01] BB MSG STATE CHANGE: FAIL
  + daemon.info CCD: [LSM:00:01] LOGIN STATE CHANGE: Wait for Login Info
  + daemon.info CCD: [TSM:03:01] TERM STATE CHANGE: \*\* TX Enable \*\*
* daemon.info CCD: [TSM:00:01] TERM STATE CHANGE: Non-Valid

## AL GUI

The ALGUI routines use Selenium libraries to automate the browser actions needed to control the ArcLight NMS, EMS and Terminal GUI’s per the map bundle test procedure [REF-23-02].

#### Pseudo-Code

[TBR-Enhance pseudo-code to include more viasatSelenium calls]

class algElements(viasatSelenium):

\_\_init\_\_(self, resultsDir):

viasatSelenium.\_\_init\_\_(“chrome”, resultsDir)

class algNms(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

algElements.\_\_init\_\_(resultsDir)

self.url = url

self.usern = user

self.passw = pw

self.resultsDir = resultsDir

class algEms(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

algElements.\_\_init\_\_(resultsDir)

self.url = url

self.usern = user

self.passw = pw

self.resultsDir = resultsDir

class algTerm(algElements):

def \_\_init\_\_(self, url, user, pw, resultsDir):

algElements.\_\_init\_\_(resultsDir)

self.url = url

self.usern = user

self.passw = pw

self.resultsDir = resultsDir

## ViaSat Selenium

The ViaSat Selenium library (VSEL) is derived (copied) from the Surfbeam Automation Manager code base file //Broadband/TestTools/SAM/core/source/lib/Selenium2.py along with member function extensions.

#### Pseudo-Code

from Selenium2 IMPORT \*

class vselenium(Selenium2):

def \_\_init\_\_(self, browserType, resultsDir):

Selenium2.\_\_init\_\_(self, "chrome", resultsDir)

def timestamp(self):

return datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d-%H-%M-%S')

def clickButton(self, TEXT):

self.clickElement (“xpath=”+TEXT):

def tearDown(self):

self.driver.quit()

## AL Devices

Update the existing TerminalSerialCom.py located in perforce at

//Arclight/ArcLight/AcceptanceTest/Automation/GTAF\_TIDE/Scripts/utils/TermSerialCom.py to support additional member functions, e.g. setAntenna()

As an interim solution, //Arclight/ArcLight/AcceptanceTest/Automation/MapTesting/src/vsel/termLinkSSH.py was created to support the SSH path and in the future should be merged with TermSerialCom.py to separate the common code (and remain backward compatable).

Please enter menu item: I

Antenna Type Definitions

1: Rantec 11.5 Antenna

2: Rantec 11.5 Narrow-fit Antenna

3: ViaSat VR12 Antenna

4: ViaSat VR12 HiPwr(10W) Antenna

5: ViaSat VR12HS HiPwr(10W) Antenna

6: ViaSat VR12Ku-HP HiPwr(20W) LO-Sel Antenna

7: ViaSat VR12 Ka-Band Antenna

8: ViaSat KuKarray Antenna

10: ViaSat HMSA Ka-Band Antenna

11: KVH V7 Antenna

12: KVH V3 Antenna

13: KVH V11 Antenna

21: Tracstar Antenna

31: RaySat Antenna

51: TECOM KuStream 1500 Antenna

52: TECOM KuStream 1015 Antenna

99: Undefined Antenna

#### Pseudo Code

@defer.inlineCallbacks

def setAntenna(self, antennaType)

"""Set the Antenna Type of the terminal image

antennaType - id of antenna, e.g. 7 ViaSat VR12 Ka-Band Antenna

"""

yield self.login()

yield self.termcfg()

# enter General Parameters menu

yield Cmd(self.tl, '0', self.resp\_termcfgPrompt).send()

# select the Antenna Type menu

yield Cmd(self.tl, 'I', 'x.xx\):').send()

yield Cmd(self.tl, antennaType, 'menu item: ').send()

# exit the Antenna menu

yield Cmd(self.tl, 'x', 'menu item: ').send()

# exit termcfg

yield Cmd(self.tl, 'x', '\(y/n\)\?').send()

# save the settings

yield Cmd(self.tl, 'y', '\(y/n\)\?').send()

# don't reboot the VMT

yield Cmd(self.tl, 'n', self.resp\_shellPrompt).send()

defer.returnValue(True)

## GTAF Devices

[TBR] Not implemented in initial releases; for future releases:

* Refactor portions from Terminal Console Interface
* Port existing IXIA code.
* Add additional common devices.

## Results Database

The ReDa API can be converted from the existing TCL package at //Broadband/TestTools/SAM/core/source/lib/RedaLibrary.tcl

The ReDa screens are installed and “working.” The following additional work is required,

* Align perforce.sql containing the ReDa MySQL definitions to those expressed in the ReDa .php webpage files.
  + This will be a reverse engineering effort since this file //Broadband/TestTools/Reda/performance/source/setup/performance.sql is out-of-date
  + GroupInfo table columns have been brought up-to-date
* Add more status and test info calls in the scipts so that the database contains status used by ReDa screens.
  + Status was modified such that running/complete is shown on the ReDa status

The ReDa screens may be found in perforce at //Arclight/ArcLight/AcceptanceTest/Automation/MapTesting/src/redaweb which is derived from a snapshot of //Broadband/TestTools/Reda/SB2/, the more generic at //Broadband/TestTools/Reda/performance/source/webpages/ are out-of-date.

PHP installation on the automation server, described in Appendix A 7.2A.10 PHP, is a prerequisite for executing ReDa webpages.

### Pseudo-Code

## MySQL

This section defines the specific details of the database table definitions. The database tables are created by calling the SQL script once.

Add the following special table constraints to simplify coding:

* Set unique keys with “ALTER TABLE <table> ADD UNIQUE <column>”

### Pseudo-Code

The SQL from //Broadband/TestTools/ReDa/performance/source/setup/performance.sql with the tables to be altered highlighted as per

Figure 17 - Database Tables

[TBR - Add ArcLight Configuration/Status tables]

-- MySQL dump 10.10

--

-- Host: localhost Database: performance

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

-- Server version 5.0.18

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Current Database: `performance`

--

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `performance` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;

USE `performance`;

--

-- Table structure for table `Browser`

--

DROP TABLE IF EXISTS `Browser`;

CREATE TABLE `Browser` (

`BrowserID` int(11) NOT NULL auto\_increment,

`Browser` varchar(40) NOT NULL,

`Version` varchar(20) default NULL,

`Connections` int(11) default NULL,

PRIMARY KEY (`BrowserID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `Browser`

--

/\*!40000 ALTER TABLE `Browser` DISABLE KEYS \*/;

LOCK TABLES `Browser` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `Browser` ENABLE KEYS \*/;

--

-- Table structure for table `CPE`

--

DROP TABLE IF EXISTS `CPE`;

CREATE TABLE `CPE` (

`CpeID` int(11) NOT NULL auto\_increment,

`OS` varchar(40) default NULL,

`RAM` int(11) default NULL,

`MAC` varchar(20) default NULL,

`CPU` varchar(30) default NULL,

`Brand` varchar(40) default NULL,

`Description` text,

PRIMARY KEY (`CpeID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `CPE`

--

/\*!40000 ALTER TABLE `CPE` DISABLE KEYS \*/;

LOCK TABLES `CPE` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `CPE` ENABLE KEYS \*/;

--

-- Table structure for table `CpeTimeSeries`

--

DROP TABLE IF EXISTS `CpeTimeSeries`;

CREATE TABLE `CpeTimeSeries` (

`Datetime` datetime NOT NULL,

`CpuUtilization` float(5,2) default NULL,

`RamUtilization` float(5,2) default NULL,

`GrpID` int(11) NOT NULL,

PRIMARY KEY (`Datetime`,`GrpID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `CpeTimeSeries`

--

/\*!40000 ALTER TABLE `CpeTimeSeries` DISABLE KEYS \*/;

LOCK TABLES `CpeTimeSeries` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `CpeTimeSeries` ENABLE KEYS \*/;

--

-- Table structure for table `GroupInfo`

--

DROP TABLE IF EXISTS `GroupInfo`;

CREATE TABLE `GroupInfo` (

`GrpID` int(11) NOT NULL auto\_increment,

`Results` varchar(60) default NULL,

`Datetime` datetime default NULL,

`DurationHrs` float(5,2) default NULL,

`GroupName` varchar(40) default NULL,

`StationName` varchar(40) default NULL,

`StationType` varchar(40) default NULL,

`User` varchar(40) default NULL,

`TestCase` varchar(80) default NULL,

`TestTool` varchar(40) default NULL,

`Product` varchar(40) default NULL,

`CpeID` int(11) default NULL,

`BrowserID` int(11) default NULL,

`Status` enum('complete','running','invalid') default NULL,

`SpecPassFail` enum('pass', 'fail') default NULL,

`Description` text,

`Notes` text,

PRIMARY KEY (`GrpID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `GroupInfo`

--

/\*!40000 ALTER TABLE `GroupInfo` DISABLE KEYS \*/;

LOCK TABLES `GroupInfo` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `GroupInfo` ENABLE KEYS \*/;

--

-- Table structure for table `ObjectTime`

--

DROP TABLE IF EXISTS `ObjectTime`;

CREATE TABLE `ObjectTime` (

`ObjID` int(11) NOT NULL auto\_increment,

`PageID` int(11) NOT NULL,

`Type` varchar(40) default NULL,

`ObjURL` varchar(256) default NULL,

`ObjSize` int(11) default NULL,

`ObjHttpResponse` int(11) default NULL,

PRIMARY KEY (`ObjID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `ObjectTime`

--

/\*!40000 ALTER TABLE `ObjectTime` DISABLE KEYS \*/;

LOCK TABLES `ObjectTime` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `ObjectTime` ENABLE KEYS \*/;

--

-- Table structure for table `PageTime`

--

DROP TABLE IF EXISTS `PageTime`;

CREATE TABLE `PageTime` (

`PageID` int(11) NOT NULL auto\_increment,

`TestID` int(11) NOT NULL,

`Datetime` datetime default NULL,

`URL` varchar(256) default NULL,

`Size` int(11) default NULL,

`Objects` int(11) default NULL,

`HttpResponse` int(11) default NULL,

`LoadTime` int(11) default NULL,

`DownloadBytes` int(11) default NULL,

`UploadBytes` int(11) default NULL,

`AcceleratedLoadTime` int(11) default NULL,

`AcceleratedDownloadBytes` int(11) default NULL,

`AccelearatedUploadBytes` int(11) default NULL,

PRIMARY KEY (`PageID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `PageTime`

--

/\*!40000 ALTER TABLE `PageTime` DISABLE KEYS \*/;

LOCK TABLES `PageTime` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `PageTime` ENABLE KEYS \*/;

--

-- Table structure for table `PerfCheckConfig`

--

DROP TABLE IF EXISTS `PerfCheckConfig`;

CREATE TABLE `PerfCheckConfig` (

`PerfCheckID` int(11) NOT NULL auto\_increment,

`TestID` int(11) NOT NULL,

`NumReps` int(11) default NULL,

`Timeout` int(11) default NULL,

`ConnectionType` varchar(20) default NULL,

`RestartClient` tinyint(1) default NULL,

`ClearDeltaCache` tinyint(1) default NULL,

`DelayBetweenSites` int(11) default NULL,

`DownloadMaxThroughput` int(11) default NULL,

`DownloadAvgThroughput` int(11) default NULL,

`UploadMaxThroughput` int(11) default NULL,

`UploadAvgThroughput` int(11) default NULL,

`UnaccelDownloadMaxThroughput` int(11) default NULL,

`UnaccelDownloadAvgThroughput` int(11) default NULL,

`UnaccelUploadMaxThroughput` int(11) default NULL,

`UnaccelUploadAvgThroughput` int(11) default NULL,

`TxBw` int(11) default NULL,

`RxBw` int(11) default NULL,

`TxDelay` int(11) default NULL,

`RxDelay` int(11) default NULL,

`TxPacketLoss` int(11) default NULL,

`RxPacketLoss` int(11) default NULL,

PRIMARY KEY (`PerfCheckID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `PerfCheckConfig`

--

/\*!40000 ALTER TABLE `PerfCheckConfig` DISABLE KEYS \*/;

LOCK TABLES `PerfCheckConfig` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `PerfCheckConfig` ENABLE KEYS \*/;

--

-- Table structure for table `PerfCheckResults`

--

DROP TABLE IF EXISTS `PerfCheckResults`;

CREATE TABLE `PerfCheckResults` (

`PageID` int(11) NOT NULL auto\_increment,

`PerfCheckID` int(11) NOT NULL,

`URL` varchar(256) NOT NULL,

`DownloadTime` int(11) NOT NULL,

`UnaccelDownloadTime` int(11) NOT NULL,

`DownloadBytes` int(11) NOT NULL,

`UnaccelDownloadBytes` int(11) NOT NULL,

`UploadBytes` int(11) NOT NULL,

`UnaccelUploadBytes` int(11) NOT NULL,

PRIMARY KEY (`PageID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `PerfCheckResults`

--

/\*!40000 ALTER TABLE `PerfCheckResults` DISABLE KEYS \*/;

LOCK TABLES `PerfCheckResults` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `PerfCheckResults` ENABLE KEYS \*/;

--

-- Table structure for table `PerfCheckSiteInfo`

--

DROP TABLE IF EXISTS `PerfCheckSiteInfo`;

CREATE TABLE `PerfCheckSiteInfo` (

`PageID` int(11) NOT NULL auto\_increment,

`PerfCheckID` int(11) NOT NULL,

`URL` varchar(256) NOT NULL,

`AccelenetTime` int(11) NOT NULL,

`UnacceleratedTime` int(11) NOT NULL,

`AccelenetDownloadBytes` int(11) NOT NULL,

`UnacceleratedDownloadBytes` int(11) NOT NULL,

`AccelenetUploadBytes` int(11) NOT NULL,

`UnacceleratedUploadByes` int(11) NOT NULL,

`Acceleration` int(11) default NULL,

`DownloadCompressionRatio` int(11) default NULL,

`UploadCompressionRatio` int(11) default NULL,

PRIMARY KEY (`PageID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `PerfCheckSiteInfo`

--

/\*!40000 ALTER TABLE `PerfCheckSiteInfo` DISABLE KEYS \*/;

LOCK TABLES `PerfCheckSiteInfo` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `PerfCheckSiteInfo` ENABLE KEYS \*/;

--

-- Table structure for table `PerfCheckTestSegnmentInfo`

--

DROP TABLE IF EXISTS `PerfCheckTestSegnmentInfo`;

CREATE TABLE `PerfCheckTestSegnmentInfo` (

`PerfCheckID` int(11) NOT NULL auto\_increment,

`TestSegmentName` varchar(80) default NULL,

`SiteList` varchar(256) default NULL,

`NumReps` int(11) default NULL,

`CompletedReps` int(11) default NULL,

`Timeout` int(11) default NULL,

`BrowserWaitTime` int(11) default NULL,

`ConnectionType` varchar(256) default NULL,

`RestartClient` tinyint(1) default NULL,

`ClearDeltaCache` tinyint(1) default NULL,

`ClearDnsCache` tinyint(1) default NULL,

`NoAcceleNet` tinyint(1) default NULL,

`DelayBetweenSites` int(11) default NULL,

`DownloadMaxThroughput` int(11) default NULL,

`DownloadAvgThroughput` int(11) default NULL,

`UploadMaxThroughput` int(11) default NULL,

`UploadAvgThroughput` int(11) default NULL,

`UnaccelDownloadMaxThroughput` int(11) default NULL,

`UnaccelDownloadAvgThroughput` int(11) default NULL,

`UnaccelUploadMaxThroughput` int(11) default NULL,

`UnaccelUploadAvgThroughput` int(11) default NULL,

`UploadBandwidth` int(11) default NULL,

`Downloadbandwidth` int(11) default NULL,

`RTT` int(11) default NULL,

`PacketLoss` int(11) default NULL,

PRIMARY KEY (`PerfCheckID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `PerfCheckTestSegnmentInfo`

--

/\*!40000 ALTER TABLE `PerfCheckTestSegnmentInfo` DISABLE KEYS \*/;

LOCK TABLES `PerfCheckTestSegnmentInfo` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `PerfCheckTestSegnmentInfo` ENABLE KEYS \*/;

--

-- Table structure for table `TestInfo`

--

DROP TABLE IF EXISTS `TestInfo`;

CREATE TABLE `TestInfo` (

`TestID` int(11) NOT NULL,

`GrpID` int(11) NOT NULL,

`Status` enum('complete','running','invalid') default NULL,

`Datetime` datetime default NULL,

`DurationHrs` float(5,2) default NULL,

PRIMARY KEY (`TestID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `TestInfo`

--

/\*!40000 ALTER TABLE `TestInfo` DISABLE KEYS \*/;

LOCK TABLES `TestInfo` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `TestInfo` ENABLE KEYS \*/;

--

-- Table structure for table `Webtimer`

--

DROP TABLE IF EXISTS `Webtimer`;

CREATE TABLE `Webtimer` (

`PageTimeID` int(11) NOT NULL auto\_increment,

`TestID` int(11) NOT NULL,

`DwellTime` int(11) default NULL,

`PageTimeout` int(11) default NULL,

PRIMARY KEY (`PageTimeID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `Webtimer`

--

/\*!40000 ALTER TABLE `Webtimer` DISABLE KEYS \*/;

LOCK TABLES `Webtimer` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `Webtimer` ENABLE KEYS \*/;

--

-- Table structure for table `queries`

--

DROP TABLE IF EXISTS `queries`;

CREATE TABLE `queries` (

`ID` int(11) NOT NULL auto\_increment,

`query` text NOT NULL,

`description` text NOT NULL,

`times\_used` int(11) NOT NULL,

PRIMARY KEY (`ID`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--

-- Dumping data for table `queries`

--

/\*!40000 ALTER TABLE `queries` DISABLE KEYS \*/;

LOCK TABLES `queries` WRITE;

UNLOCK TABLES;

/\*!40000 ALTER TABLE `queries` ENABLE KEYS \*/;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

# Requirements traceability

[This section shall contain:

a. Traceability from each software unit identified in this SDD to the CSCI requirements allocated to it. (Alternatively, this traceability may be provided in 4.1.)   
b. Traceability from each CSCI requirement to the software units to which it is allocated.]

[This section provides the traceability between each software component identified in this SDD and the corresponding requirements for those components as defined within the [CSCI Name] SRS. That traceability is provided 1) from each software component TO the requirements that have been allocated to it and 2) from each requirement TO the software component(s) to which it is allocated. Example is provided below, it may be preferable to use an appendix or external reference]]

| **Component Allocation** | **Device SRS DOORS Paragraph Number** | **Requirement ID** |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Software Component to Requirement mapping

| **Requirement ID** | **Device SRS DOORS**  **Paragraph Number** | **Component Allocation** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Requirement to Software Component Mapping

# Notes

This section shall contain any general information that aids in understanding this document.

## Definitions

This section contains common definitions that apply to terminology used within this document.

## Acronyms

This section defines all acronyms used within this document.

|  |  |  |
| --- | --- | --- |
| **Acronym** | **Definition** | **Comment** |
| CSCIs | Computer Software Configuration Items |  |
| ECS | Enhanced Channel Simulator | Used to simulate satellite delay |
| EFLM | Enhanced Forward Link Modulator | Used for VCSM Forward Link |
| EHC | Enhanced Hub Canceller | Enhanced Hub Canceller used for PCMA |
| GTAF | Generic Test Automation Facility | In-house libraries for controlling test devices, i.e. IXIA |
| HWCIs | Hardware Configuration Items |  |
| IPE | IP Encapsulator | Used to encapsulate IP packets |
| RTNMS | Real-Time Network Management Server | Used to control and manage remote terminals |
| VMBR | ViaSat Mobile Broadband Router | Remote Terminal used in ArcLight Hub |

Appendix A - Automation Server Configuration

This appendix describes additional configuration information for the Automation Server not found in the Apache and MySQL sections of this document.

GNOME

GNOME [optional] needs to be installed to provide a Console GUI; follow instructions at

http://unix.stackexchange.com/questions/181503/how-to-install-desktop-environments-on-centos-7

Python

Python comes installed on CentOS 7, but “pip” also needs to be installed. See <https://pip.pypa.io/en/stable/installing.html> to download get-pip.py and install on CentOS target using: python get-pip.py.

Selenium

Python API’s for Selenium

Chrome and Chromedriver

Install Chrome and Chromedriver; requires the Richard Llyod Chrome install for CentOS 7 since Chrome no longer supported by CentOS.

export PATH=$PATH:/tmp/automation\_addons (add to install from better location)

export LD\_LIBRARY\_PATH=/opt/google/chrome/lib

Xvfb for Headless Display

Run a virtual display service in the background for the Selenium/WebDriver Browser

Installation:

yum install xorg-x11-server-Xvfb

Shell based launch:

Xvfb :47 -ac -screen 0 1280x1024x24 &

Add File:

[/etc/systemd/system/xvfb@.service](mailto:/etc/systemd/system/xvfb@.service)

Contents:

[Unit]

Description=virtual frame buffer X server for display %I

After=network.target

[Service]

ExecStart=/usr/bin/Xvfb %I -ac -screen 0 1280x1024x24

[Install]

WantedBy=multi-user.target

Start services in linux:

%systemctl enable [xvfb@:47.service](mailto:xvfb@:47.service)

%systemctl start [xvfb@:47.service](mailto:xvfb@:47.service)

x11vnc for Monitoring Headless Display

Run a vnc service in the background to connect a RealVNC/TightVNC client (on port 5900) and observe the Selenium GUI automation

Installation:

Native

Shell based launch:

x11vnc -rfbport 5900 -display :47 -forever -shared

Add File:

[/etc/systemd/system/x11vnc@.service](mailto:/etc/systemd/system/x11vnc@.service)

Contents:

[Unit]

Description=x11vnc server for xvfb display %I

[After=xvfb@.service](mailto:After=xvfb@.service)

[Service]

Type=simple

ExecStart=/usr/bin/x11vnc -rfbport 5900 -display %I -forever -shared

[Install]

WantedBy=multi-user.target

Start services in linux:

%systemctl enable x11vnc@:47.service

%systemctl start x11vnc@:47.service

MATLAB for Linux

MATLAB Development Tool

* Version supported by ViaSat License Server
  + R2014a(8.3)
  + file://dc1nas\Software\Applications\MATLAB\R2014a(8.3)/Linux\_Mac/
* Installation instructions from ViaSat IT
  + <file://dc1nas/Software/Applications/MATLAB/R2014a(8.3)/Linux_Mac/MATLAB_Installation_2014a(8.3).doc>
  + file://dc1nas/Software/Applications/MATLAB/R2014a(8.3)/Linux\_Mac/install\_guide.pdf
* Wiki Info
  + <https://wiki.viasat.com/display/Engineering/MATLAB+Tool>
* Runtime Compiler installation
  + unzip /usr/local/MATLAB/R2014a/toolbox/compiler/deploy/glnxa64/ MCRInstaller.zip
* Other
  + Standalone
    - <http://www.bu.edu/tech/support/research/software-and-programming/common-languages/matlab/standalone/>
  + License Query
    - mode con cols=120 lines=1000
    - cmd.exe /k "\\vianas\Projects-EF\Engineering\_Tools\License\_Query\lmutil" lmstat -a -c 27000@vcalic07

MATLAB Compiled Runtime (MCR) Libraries

Instructions to Load MATLAB Compiled Runtime libraries on Linux server; no license required, specifies version matching the MATLAB used to develop/compile:

* Download the Linux 64-bit version of the MCR for R2014a (8.3) from the MathWorks website: <http://www.mathworks.com/products/compiler/mcr/index.html>
* Unzip the file on the linux server where the application or component will be run
* Run the ./install file as root
* Click through the MathWorks installation wizard until you get to the “Folder Selection” screen
  + Enter the <mcr\_directory> (Default/Preferred: /usr/local/MATLAB/MATLAB\_Compiler\_Runtime/v83)
* Note <mcr\_directory> listed on the “Confirmation” screen and click “Install”
* After installation, patch the libstdc++.so.6 as follows to replace libstdc++.so.6.0.17 with the system libstdc++.so.6 which should point to libstdc++.so.6.0.19 or greater (verify with system administrator).
  + cd <mcr\_directory>/sys/os/glnxa64
  + rm libstdc++.so.6
  + ln –s /usr/lib64/libstdc++.so.6 libstdc++.so.6
* Example: ./run\_XXX.sh /usr/local/MATLAB/MATLAB\_Compiler\_Runtime/v83 <argument\_list>

Other

* Perforce p4 client
  + Get latest from <http://www.perforce.com/downloads/helix#product-6>
    - Copy to /usr/local/bin
    - chmod 555
  + setup .p4rc (~arclight/perforce/mwolff/.p4rc)
    - P4USER=mwolff
    - P4PORT=10.24.37.8:3002
    - P4CLIENT=mwolff\_alAutoLinuxDev
* Python environment for Perforce tree organization (~arclight/perforce/mwolff/.bashrc)
  + #!/bin/bash
  + export PYTHONPATH=$PYTHONPATH:/home/arclight/perforce/mwolff/MapTesting/src/alam:/home/arclight/perforce/mwolff/MapTesting/src/aldb:/home/arclight/perforce/mwolff/MapTesting/src/alg:/home/arclight/perforce/mwolff/MapTesting/src/altg:/home/arclight/perforce/mwolff/MapTesting/src/mapbun:/home/arclight/perforce/mwolff/MapTesting/src/mysql:/home/arclight/perforce/mwolff/MapTesting/src/vsel:
* Renaming ip devices:
  + ip link set dev ${old interface name} name ${new interface name}
* For DNS, create /etc/resolv.conf with
  + # Generated by NetworkManager
  + search hq.corp.viasat.com
  + nameserver 10.24.5.135
* Interfaces
  + <http://baturin.org/docs/iproute2/>
  + /etc/sysconfig/network-scripts/ ifcfg-eth1.816
  + VLAN Configuration
    - ip link add name eth1.816 link eth1 type vlan id 816
    - ip address add 192.168.106.47/24 dev eth1.816
    - ip route add 192.168.106.0/24 dev eth1.816
    - ip link set dev eth1.816 up
* ssh/scp
  + Setup /etc/ssh/ssh\_known\_hosts with an entry for each terminal (supported by VLAN) and RTNMS for remote termcfg and syslog retrieval
* Console
  + Use startx to bring up console GUI on CentOS

PHP

Install PHP and phpMyAdmin for REDA

* To install PHP on CENTOS-7
  + Follow these instructions, <https://centoshelp.org/servers/web/installing-php-5-5x-or-5-6x-on-centos/>
  + yum install php56u
  + yum install php56u-mysqlnd.x86\_64
  + yum install php56u-mbstring.x86\_64
  + yum install php56u-xml.x86\_64
  + yum install php56u-xmlrpc.x86\_64
  + yum install php56u-opcache.x86\_64
  + yum install php56u.x86\_64
  + yum install php56u-bcmath.x86\_64
  + yum install php56u-fpm.x86\_64
  + yum install php56u-gd.x86\_64
  + yum install php56u-mcrypt.x86\_64
  + yum install php56u-zip.x86\_64
  + yum install php56u-session.x86\_64
* To install phpMyAdmin,
  + yum install phpmyadmin (prerequisite: epel-release-7-5.noarch (rpm -qa | grep epel)
  + Hint: http://www.krizna.com/centos/install-phpmyadmin-centos-7/

Third-level Appendix A heading

Body text in Appendix A.x.y.

Appendix B - Hub Setup

Server configuration notes

* NMS
  + SuSE 11.3
* EMS, RtNMS, IPE
  + SuSE 11.1
* General
  + Use yast to configure

Second-level Appendix B heading

Body text in Appendix B.1.

Third-level Appendix B heading

Body text in Appendix B.1.1.