Software Verification User's Guide for the Data Link Communication Application (DLCA) ARINC 661 Projects

Document Number 945-9320-500
Revision -
CAGE Code 0EFD0

Rockwell Collins

Contract Number None

	NAME	TITLE	APPROVAL
Prepared By:	Navaj R. Tirupathi	Preparer	N/A
Approved By:	Jayaprakash Anandhan	Engineering	On File
Approved By:	Eileen P. Roberson	DAC Engineer	On File

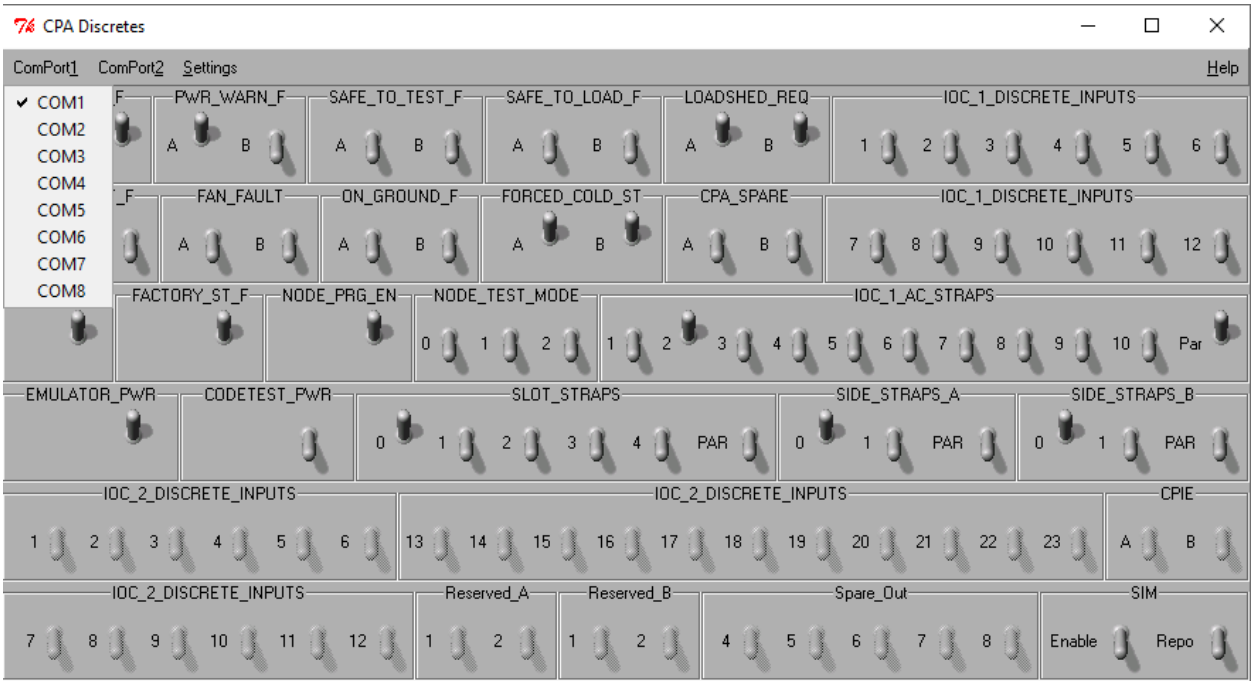
REVISION HISTORY

VER	REV	DESCRIPTION	DATE	APPROVED
001	-	Initial Release	2015-04-30	James M. Wolff
002	-	2.x Release	2017-03-01	Sridher Kamineni
300	-	3.x Release	2018-01-16	Sridher Kamineni
500	-	5.x Release	2023-03-06	Jayaprakash Anandhan

Table of Contents

List of Tables

List of Figures



1 Scope

1.1 Purpose

The purpose of this document is to capture all verification day-to-day activities and list all verification related artifacts. This document contains all details associated with the project's verification effort. The details in this document include enough information for training, for consistency and repeatability of verification activities and artifact preparation, and for understanding why the activities and artifacts are in place to assist with continuous process improvement. All decisions made relating to a verification issue should be documented to promote communication and consistency in verification.

1.2 Applicability

This document applies to the Data Link Communication Application (DLCA) ARINC 661 projects of the Rockwell Collins Commercial Systems Data Link group.

2 Reference Documents

The documents listed in this section are referenced in one or more places throughout this document. This section provides the precise title, publisher, control numbers (if any), and date of publication (if necessary for control) of each referenced document. For easy identification, each point of reference includes a bracketed number (defined in this section) that corresponds to the document being referenced.

2.1 Rockwell Collins (RC) Internal Documents

Project-Specific Documents

[1] Software Test Procedures (STP) for the Data Link Communication Application (DLCA) ATN-FANS.

I.e. STP for DLCA6500

[2] Software Requirements Specification (SRS) for Common System Services Data Link Communication Application (DLCA), RCPN 945-0592-500 Rev -

I.e. SRS for Common System Services

[3] Software Requirements Specification (SRS) for Aeronautical Telecommunication Network (ATN) Context Management (CM) And Controller Pilot Data Link Communication (CPDLC), RCPN 945-0591-500 Rev -

I.e. SRS for Core for ATN

[4] Software Requirement Specification (SRS) for Data Link Communication Application (DLCA) Future Air Navigation System (FANS-1/A) RCPN 945-0516-500 Rev -

I.e. SRS for Core for FANS-1/A

[5] Software Requirements Specification (SRS) for Data Link Communication Application (DLCA-6500) Human Machine Interface (HMI), RCPN 945-0517-500 Rev -

I.e. SRS for DLCA-6500 HMI

[6] High Level Software Requirements Specification (SRS) for Pro Line Fusion DLCA-6500 Data Link Communications Application, RCPN 945-9216-500 Rev -

I.e. SRS for DLCA-6500 HLR

[7] Software Development Plan (SDP) for the Commercial Systems (CS) Data Link Projects, RCPN 829-6997-600 Rev J

[8] Software Verification Procedures and Results (SVPR) for the DLCA-6500, RCPN 945-0520-510 Rev –

[9] Software Design Document (SDD) for the DLCA-6500 Data Link Software Component Design ,RCPN 945-7650-500 Rev -

3 Verification Artifacts

The verification artifacts include items such as Peer Review packages, Software Test Procedures documents including Test Procedure documents and associated actual test results, Test Case Summary (TCS) tables and Structural Coverage Results.

Below is a list of some common acronyms used in this document.

Table 1: List of acronyms used in this document

Acronym	Description
A661	Arinc 661
AFD	Adaptive Flight Display
ACARS	Aircraft Communication Reporting and Addressing System
ACF	ACARS Convergence Function
ACP	Access Control Plan
ADS	Automatic Dependent Surveillance
AFN	ATS Facilities Notification
AGPS	Air/Ground Protocol Stack
ATC	Air Traffic Control
ATN	Aeronautical Telecommunication Network
CCP	Cursor Control Panel
CM	Configuration management
CPCI	Computer Program Component Index
CPDLC	Controller Pilot Data Link Communications
CPN	Collins Part Number
CR	Change Request
CS	Commercial Systems
CSS	Common System Services
DLCA	Data Link Communication Application
DOORS	Dynamic Object Oriented Requirements System
DMGR	Data Manager
EPA	Engineering Project Assistant

Acronym	Description
EPS	Engineering Project Specialist
FANS	Future Air Navigation System
FMS	Flight Management System
HMI	Human Machine Interface
JIRA	Issue tracking and project management framework
LAN	Local Area Network
MKP	Multi-Functional Keyboard Panel
PREP	Peer Review Eclipse Plug-in
RCPN	Rockwell Collins Part Number
RIU	Radio Interface Unit
SCA	Structural Coverage Analysis
SCL	Software Configuration Library
SDP	Software Development Plan
SRS	Software Requirements Specifications
STP	Software Test Procedure
SVN	Sub Version
SVPR	Software Verification Procedures and Results
TC	Test Case
TCS	Test Case Summary
VectorCAST	Code coverage Tool
VISTA	Virtual Integrated Software Testbed for Avionics
WE	WORK ELEMENT in JIRA
WP	WORK PACKAGE in JIRA

3.1 DOORS Traceability

3.1.1 Purpose

A DOORS document titled "STP for DLCA6500"[1] is used for maintaining a Traceability Matrix. This is a proxy STP module which contains a list of Test Procedures, whose DOORS objects are linked to the DLCA software requirements. The software requirements which this STP links to are found in the DOORS documents titled "SRS for Common System Services"[2], "SRS for Core for ATN"[3], "SRS for Core for FANS-1/A"[4], " SRS for DLCA-6500 HMI"[5], " SRS for DLCA-6500 HLR"[6].The link starts from Test Procedures in the STP (Out Link) and ends at the Requirements in one of the SRS's (In Link). The actual test procedures and test cases are (generally) Python scripts stored in the SVN repository at <Verification-Branch>/test_procedures which reference utilities located at <Verification-Branch>/utilities.

3.1.2 Location and Structure

DLCA Modules are located in the "DLCA ATN-FANS" Project on the Collins1 server. Below is a Screenshot of the current structure of DLCA project to access SRS and STP.

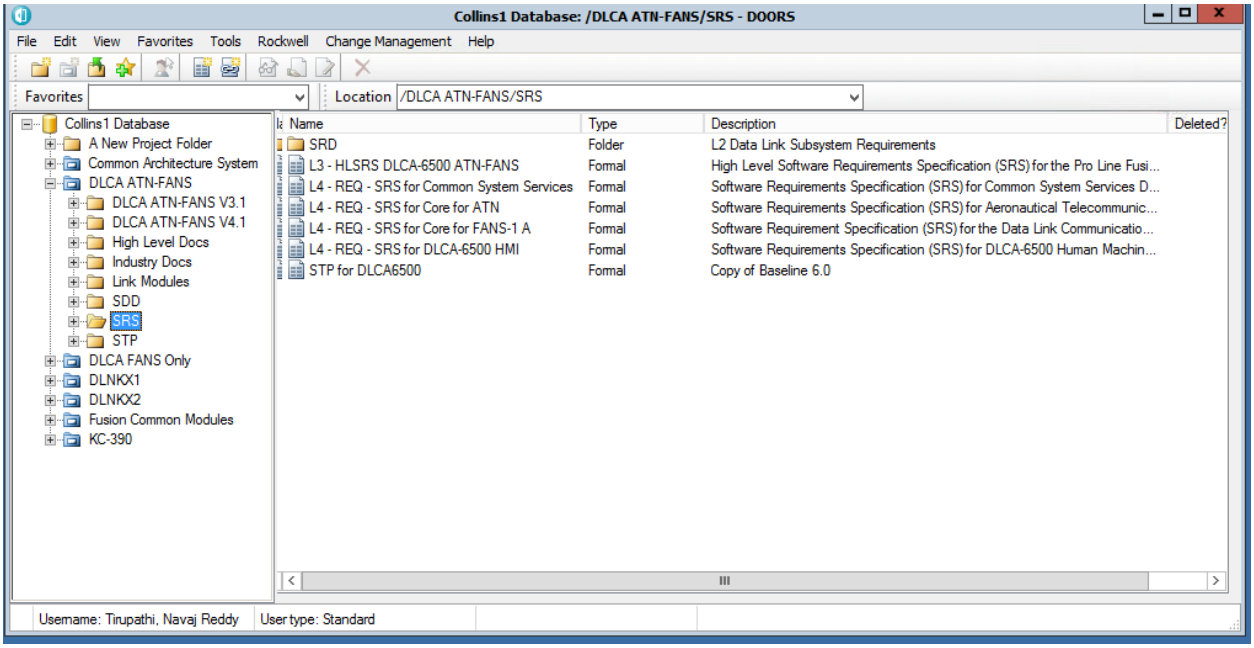


Figure 1: SRS location in DOORS

All the DLCA SRSs have already been prefixed with these suffixes.

L4 - REQ - SRS for Common System Services Prefix = **CSS**
L4 - REQ - SRS for Core for ATN Prefix = **ATN**
L4 - REQ - SRS for Core for FANS-1 A Prefix = **FANS**
L4 - REQ - SRS for DLCA-6500 HMI = **HMIX**
Note: "X" in HMIX refers to specific product

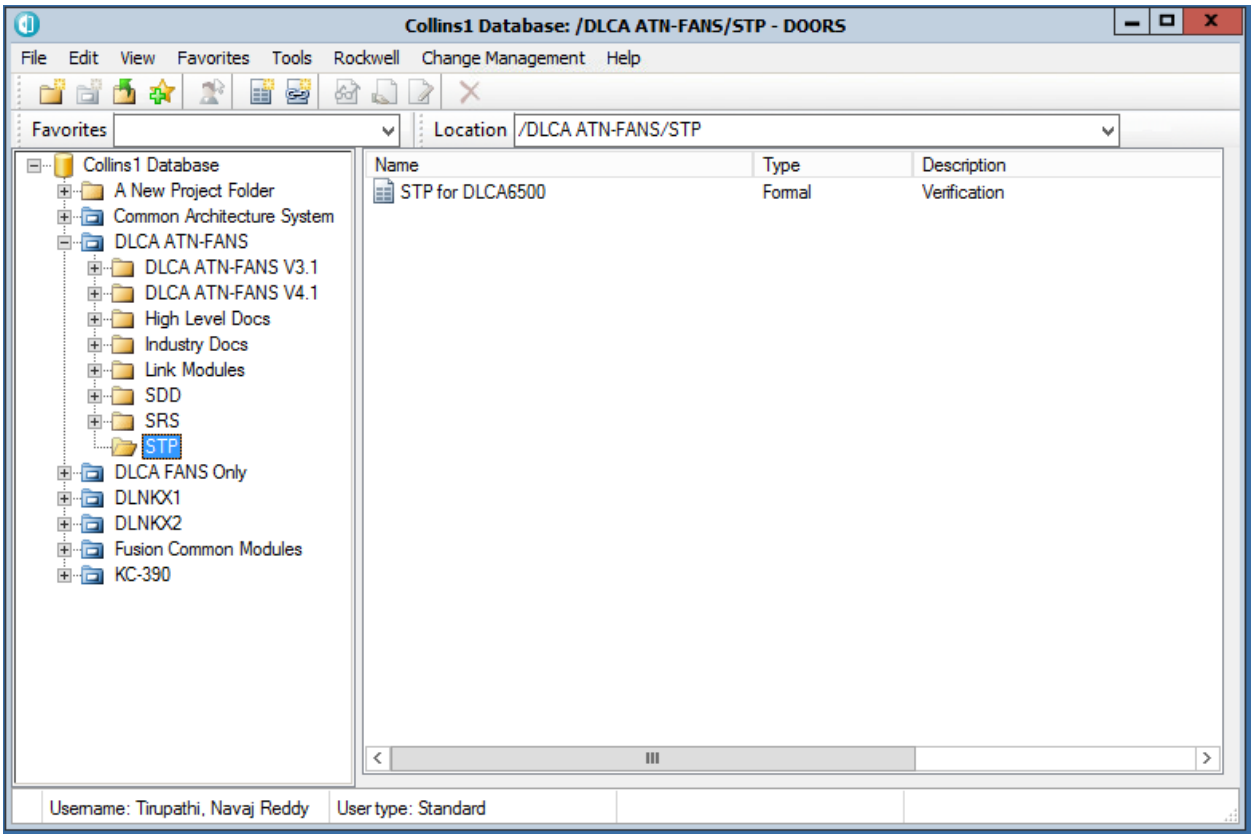


Figure 2: STP location in DOORS

Requirements will be linked at the test procedure and test case level. Section 2 of STP for DLCA document contains a listing of every test procedure created. This linking document must be updated prior to holding any review on the corresponding test case.

Each requirement should be linked to one or more test cases and test procedures, indicating that the requirement is verified completely.

DLCA Verification Traceability (STP) has the following columns

1. Object Identifier
2. DLCA Verification
3. _Verification Method
4. _AllocatedTo
5. _HMI_or_Core
6. _Comments

And below is the screenshot of STP in DOORS.

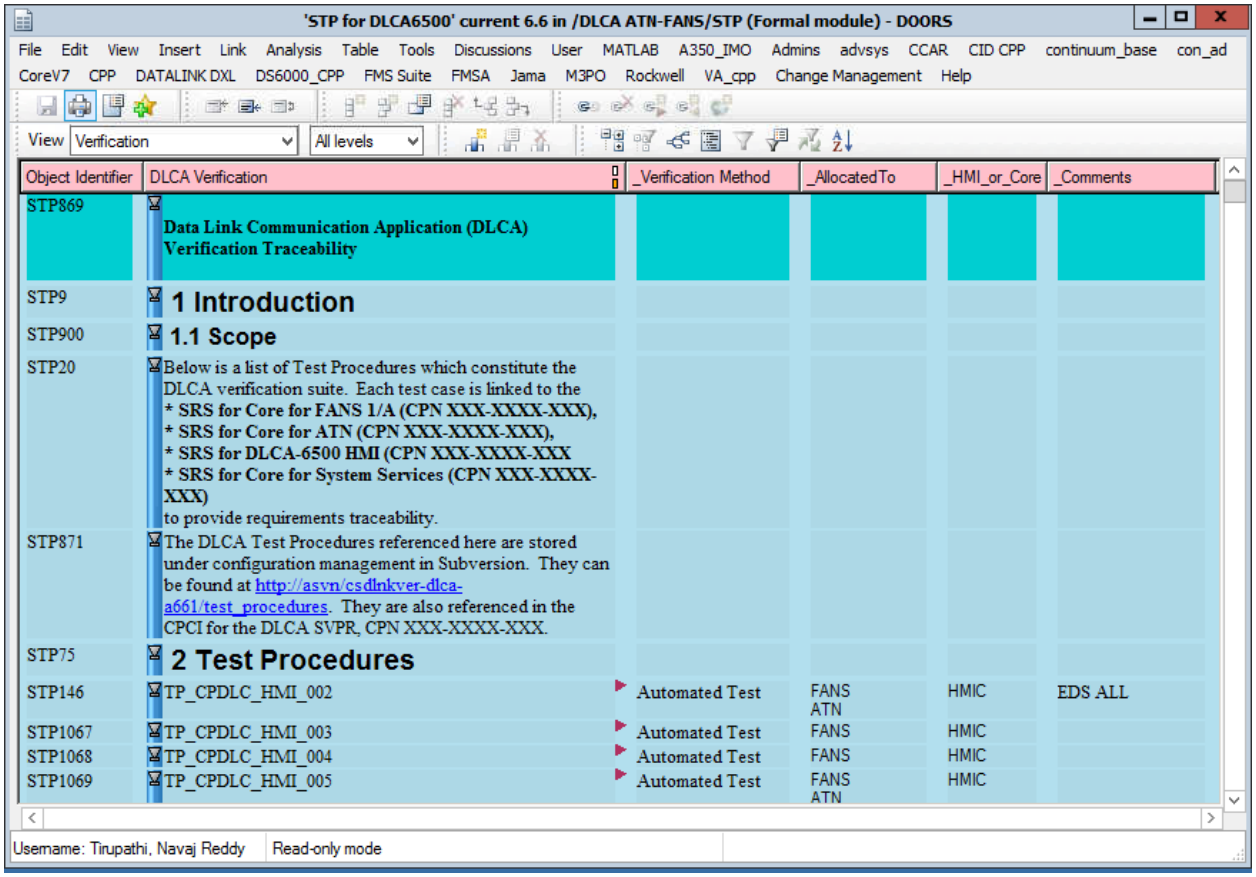


Figure 3: Example of STP in DOORS

The STP columns are detailed below:

_Test Procedure: This attribute lists the name of the test procedure, which is represented by a naming convention of TP_XXXXX_YYYY_### (e.g. TP_CPDLC_HMI_002, TP_SYSCLOCK_001, etc.).

The DLCA test procedures are stored under configuration management in Subversion. They can be found at <Verification-Branch>/test_procedures which reference utilities located at <Verification-Branch>/utilities.

_Verification Method: This attribute should indicate which verification method is performed within the test procedure. The choices are as follows:

- Automated Test – We will use this attribute for all automated tests by using automated commands
- Manual Test – We will use this attribute for all manual tests.
- Inspection – We will use inspection method only for those requirements for which an automated or manual test cannot be generated
- Analysis – We will use this attribute for algorithmic intense software requirements. This method is used when, only a visual observation of software implementation is not directly possible and no other tests methods are possible for testing the requirement.

_AllocatedTo: This attribute indicates whether a test procedure is allocated FANS or ATN or both.

_HMI or_Core: This Core attributes with values HMIC, Core

_Comments: Any remarks related to the specific test procedure should be mentioned in this attribute.

3.2 Software Test Procedures

The above-mentioned STP for DLCA ATN-FANS DOORS document identifies which test procedures verify every software requirement. From the DOORS STP document, a verifier can locate the actual test procedures and cases in SVN at <Verification-Branch>/test_procedures which reference utilities located at <Verification-Branch>/utilities.

Note: <Verification-Branch> in this document refers to: for verification development activities: https://alasvn/csdlnkver-dlca-a661/branches/5.0.0_Ver

Note: While execution of any script, we need to select specific configuration required for that script under the section precondition. Refer Utilities for more details.

3.2.1 Python Scripts

Test drivers written in Python are used to derive the verification result for requirement based testing on host and target environment. Verification cases are defined based on an analysis of software requirements. Requirement-based testing is used to show that the software performs its intended function and that the software meets its requirements. Formal host-based testing may be used for obtaining structural coverage.

3.2.1.1 Description and Layout of a Script

3.2.1.1.1 Template of TP_ZZ_XXXXX_YYY_###.py

```
testProcTitle = """
#+------------------------------------------------------------------------------+
#|                          Copyright 20XX-20YY Rockwell Collins                |
#|                              All Rights Reserved                             |
#|                            Proprietary Information                           |
#+------------------------------------------------------------------------------+
#
# $Revision:  $
# $Date:  $
# $Author:  $
# $HeadURL:  $
#
#+------------------------------------------------------------------------------+
#|DESCRIPTION:                                                                  |
#+------------------------------------------------------------------------------+
#|This script verifies XXXXXXXXXXXX XXXXXXXXXXXXX                               |
#+------------------------------------------------------------------------------+
#|REQUIREMENTS:                                                                 |
#+------------------------------------------------------------------------------+
#|  HMIC : xxxxx, xxxxx, xxxxx                                                  |
#|  CSS   : N/A                                                                 |
#|  FANS  : N/A                                                                 |
#|  ATN   : N/A                                                                 |
#+------------------------------------------------------------------------------+
"""
#+------------------------------------------------------------------------------+
#|PRECONDITION:                                                                 |
```

```
#+------------------------------------------------------------------------------+
#| * Set the XXXXXXXXXXXX XXXXXXXXXXXXX.                                        |
#| * Set the XXXXXXXXXXXX XXXXXXXXXXXXX.                                        |
#+------------------------------------------------------------------------------+
#| NOTE :                                                                       |
#+------------------------------------------------------------------------------+

# Import the necessary python libraries

# Get the Current directory of test script
currentDir = os.path.join(os.path.dirname(inspect.currentframe().f_code.co_filename))
# Execute Startup script

# Import necessary utility files

# Execute widget mapping utility to localize all widgets

# Create Logger

# Log the test case header
logger.infoWrap(testProcTitle)

testCaseDescrip = """
#+==============================================================================+
#|==============================================================================|
#|                                                                              |
#|TEST CASE    : TC_ZZ_XXXXX_YYY_###-01                                         |
#|                                                                              |
#|==============================================================================|
#+==============================================================================+
#|REQUIREMENTS     : HMICXXXX, HMICXXXX                                         |
#|PURPOSE          : To verify                                                  |
#|                   [XXXXXXXXX] XXXX XXXXXXX XXXXXXX XXXXX                     |
#|                   [XXXXXXXXX] XXXX XXXXXXX XXXXXXX XXXXX                     |
#|TEST TYPE        : Normal/Boundary/Out of Range Testing                       |
#|CONDITION        : XXXXX XXXXXXXX XXXXXXX XXXXXXXX                            |
#|NOTE             :                                                            |
#+-----+-----------------------------+----------------------------------------+
#|STEPS|       INPUT                 | EXPECTED RESULTS                       |
#|-----+-----------------------------+----------------------------------------+
#|     |                             |                                        |
#+------------------------------------------------------------------------------+
#|Pass/Fail criteria: Tests will be considered PASSED if the actual matches     |
#|                    the expected.                                             |
```

```
#+------------------------------------------------------------------------------+
"""
logger.infoWrap(testCaseDescrip)
# Set test case
logger.setTestCase("TC_ZZ_XXXXX_YYY_###-01")

#+------------------------------------------------------------------------------+
#|START OF PROCEDURE: TC_ZZ_XXXXX_YYY_###-01                                    |
#+------------------------------------------------------------------------------+
#
#+------------------------------------------------------------------------------+
#|END OF PROCEDURE: TC_ZZ_XXXXX_YYY_###-01                                      |
#+------------------------------------------------------------------------------+
#
#
testCaseDescrip = """
#+==============================================================================+
#|==============================================================================|
#|                                                                              |
#|TEST CASE    : TC_ZZ_XXXXX_YYY_###-02                                         |
#|                                                                              |
#|==============================================================================|
#+==============================================================================+
#|REQUIREMENTS     : HMICXXXX, HMICXXXX                                         |
#|PURPOSE          : To verify XXXX XXXXXXX XXXXXXX XXXXX                       |
#|TEST TYPE        : Normal/Boundary/Out of Range Testing                       |
#|CONDITION        : XXXXX XXXXXXXX XXXXXXX XXXXXXXX                            |
#|NOTE             :                                                            |
#+-----+------------------------------+---------------------------------------+
#|STEPS|       INPUT                  | EXPECTED RESULTS                      |
#|-----+------------------------------+---------------------------------------+
#|     |                              |                                       |
#+------------------------------------------------------------------------------+
#|Pass/Fail criteria: Tests will be considered PASSED if the actual matches     |
#|                    the expected.                                             |
#+------------------------------------------------------------------------------+
"""
logger.infoWrap(testCaseDescrip)
# Set test case
logger.setTestCase("TC_ZZ_XXXXX_YYY_0XX-02")

#+------------------------------------------------------------------------------+
#|START OF PROCEDURE: TC_ZZ_XXXXX_YYY_###-02                                    |
#+------------------------------------------------------------------------------+
```

```
#
#+------------------------------------------------------------------------------+
#|END OF PROCEDURE: TC_ZZ_XXXXX_YYY_###-02                                      |
#+------------------------------------------------------------------------------+
#|                                                                              |
# Close the logger                                                              |
logger.close()                                                                  |
#|                                                                              |
# Reset the Environment                                                         |
util_Test_Support.dlcaClose()                                                   |
#|                                                                              |
#+--------------------- End of Test Procedure --------------------------------+
```

Reference: Section 4.2 explains about the terminologies of Test Procedure Identifiers

In the above mentioned Layout of Python Test script contains the following sections:

- *Section 1* - Rockwell Collins Proprietary Information is provided (Copyright information is not a mandatory field)
- *Section 2* - Latest Revision information and information on the revision history of the file is provided.
- *Section 3* – Setup steps to execute DLCA startup script and importing necessary utilities.
- *Section 4* - Define all test cases for the test procedure.
- *Section 5* – Shutdown VISTA and close the Log when execution is completed.

The test procedure should contain the following:

Rockwell Collins Proprietary Information

Description

This section contains the description of all the requirements covered in the entire TP

Requirements Covered

This contains a list of COMMON SYSTEM SERVICES, ATN, FANS and HMI requirements verified in this Test Case.

Test Case ID

logger.setTestCase("TC_ZZ_XXXX_YYYY_###-##")

This is a VISTA command used to define the test case.

Test Cases

This section contains the overview of requirements, purpose, inputs, conditions, expected results verified in this test case.

Requirements

This contains a list of requirements which are logically grouped as per the functionality.

Purpose

The purpose of test cases needs to be mentioned.

TEST TYPE

Contains the type of the test. Like Normal/Robust/Boundary/Out of Range Testing/Equivalence class

INPUT

Contains the list of input actions performed in test case.

Condition

The conditions which are required to meet the expected results.

EXPECTED RESULT

Contains expected result for each requirement, verified in test case.

3.2.1.1.2 Test Case Templates

Core Testing

This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/Core_Test_Case_Example.py

HMI Page Testing

This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/HMI_Page_Testing_Example.py

HMI Range Testing

This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661//HMI_Range_Testing_Example1.py

Code Inspection

This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/Code_Inspection_Example.py

Codec Testing

For ATN: This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/Codec_Test_Case_Example_ATN.txt

For FANS: This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/Codec_Test_Case_Example_FANS.txt

Other Entries

As the project progresses we will add templates for all other entry types: {Altitude, Speed, Temperature, Time, Position, Transition, Procedure, Distance, Soul, Offset, Beacon code, Winds, Place Bearing Distance, Facility Designator, Facility Name}

Note: TP templates for all projects will be same but the prefixes will be changed according to the requirement section in respective SRS.

How to test Untestable functionality

Occasionally, builds are delivered in which a functional area cannot be reached, or a test procedure cannot be written to test a certain function. This test procedure must be corrected using a subsequent released build. In order to flag the failure, use the Vista python utility function: logger.testFailed, for example:

```
logger.testFailed("Gnd App Warning Page Not accessible.")
```

HMI Requirement "Page shall set the Page fields to the default state:"

This phrase indicates that the page we have entered has some default format. This format may be in FANS mode or ATN mode or both. We will use the run time A661 Trace

to compare the fields of the page with the state of the fields defined in the requirement. The real time

A661 trace is obtained from the A661Parser Wrapper functions located at <Verification-Branch>/utilities/libs.

The below defined function helps us to get the real time A661 trace of the widget. It considers

Widget, Parameter ID, WKN_WKS as parameters.

A661Parser.find(widget=data['widget'], parameterId=eachRMP, wkn_wks=wkn_wks)

The A661 based wrapper functions are being called from utilities files located at

/utilities/util_Test_Support.py

Ex: TPs use the following notation is used to check whether the send function is enabled or not.

by getting the real time trace.

logger.testWrap(util_Test_Support.findWidgetEvent(trace=True,Widget=SEND_FUNCTION,parameter=[WIDGET_ENABLE],instance="L_AFD_DLCA")==[True])

findWidgetEvent utility internally calls the A661Parser.find function to acquire the real time trace

We will compare the LHS result with the RHS Value [True].

If the LHS side value is [True], We consider the function is enabled and Test is considered as pass

If the LHS side value is [False], We consider the function is disabled and Test is considered as Fail

If the LHS side value is ["WIDGET NOT FOUND"], We consider the A661 trace of the function is not found and test is considered as Fail

Logger.testWrap compares the result with the RHS value and logs the result into the log file.

3.2.1.2 Test Script Writing Standards

- The 'timeout=' qualifier should only be specified when there is a specific timing need where the default (10 seconds) is not acceptable.
- Hard coded wait values are not to be used with the 'wait', 'runUntilWrap/timeout' commands unless absolutely necessary and the reason is documented in the test. Instead, a defined constant shall be used.
- When entering data from the MKP into a data field on the display, sufficient check(s) should be performed to ensure the data was actually entered into the data field. For example, when entering 1234 from the scratch pad into the Altitude data field, a runUntilWrap command could be used to ensure the data actually gets entered into the data field.
- When using the runUntilWrap commands, specify a period to indicate how often the command should test for the condition. The default is period=0.5 second (every other cycle). If a higher period is acceptable, it should be specified.
- A runUntilWrap shall always be used if a requirement does not specify a behavior to happen immediately. For example, if a SEND is expected to be enabled, but there is no timing requirement which indicates how quickly the SEND should enable, the runUntilWrap should be used while checking for the SEND. This allows for the behavior to appear in 10 seconds. Depending on the performance of the simulation, this allows a standard tolerance when one has not been explicitly identified.
- The testWrap command shall not be used outside of defined test cases in a test procedure. A test case is defined using the setTestCase command.
- The infoWrap command will be used to log any information for test case or test procedure

  Ex: logger.infoWrap(testProcTitle)
- Verify that test cases are written for 'Resolution' and 'Hysteresis' if applicable.

- Verify that test cases are written for Boundary Value analysis, Robustness, Out of Range and Equivalence Class Partitioning. These should be explicitly mentioned in the test case description or comment section.
- Naming Conventions for Uplink messages:
    - If new uplink messages need to be created for use in a test procedure (.txt files), the name of the file shall be the uplink number, followed by an underscore, followed by up to 15 characters of meaningful description (e.g. um163_ERRUNEXPDATA.txt)
- Ensure that Trailing spaces are removed and spell checks are performed.
- Ensure that any utility utilized in the test procedure is approved by the Lead Verification Engineer. Utilities shall be peer reviewed.
- Ensure that the Traceability in Doors is matching with the requirements mentioned in the Test Procedure.
- For indentation, use 4 spaces per indentation level.
- Imports shall be on separate lines, e.g.:

    import os

    import sys
- It is recommended to use Imports, at the top of the file, just after any module comments and before module globals and constants.
- Imports should be grouped in the following order:
    - standard library imports
    - local application/library specific imports
- For constants definition, use UPPERCASE.
- Variable, method and package names shall use the "mixedCase" or "mixed_Case" naming convention.
- CPN, ENVIRONMENT and Build number should be present at the start of log file.
- It could be good to follow that Test Procedure should not exceed 1000 lines

3.2.1.3 Utilities

- The Utilities are Python programs that are designed to serve as reusable "building blocks" for constructing test files, by encapsulating functions that are commonly included in more than one verification test. Use a utility whenever possible, to save time by not rewriting an existing segment of Python.
- Other Python utilities shall be used whenever possible. Recursive or circular calls are not permitted.
- Each utility shall have a standard header as per the template.

```
# +------------------------------------------------------------------------+
# |                        Copyright 20XX-20YY Rockwell Collins            |
# |                             All Rights Reserved                        |
# |                           Proprietary Information                      |
# +------------------------------------------------------------------------+
# LATEST VERSION INFORMATION
# $Revision: $
# $Date: $
```

```
# $Author: $
# $HeadURL: $
# +-------------------------------------------------------------------------------+
# Purpose:                                                                        |
# Format:                                                                         |
# Parameters:                                                                     |
# Example:                                                                        |
# +-------------------------------------------------------------------------------+
```

- If no parameters exist for a utility, the field shall be marked as 'N/A'.
- Using testWrap command in utilities is not allowed. Instead, use runUntilWrap.

The runUntilWrap command shall be used whenever possible.

- The wait command shall not be used unless it is deemed entirely necessary.

Process for Creating a New Utility:

- Create a new .py file with appropriate naming convention, which will have Rockwell Collins Copyright and Proprietary information and latest version information.
- The next section will be the purpose of creating the utility.
- Finally, write a small piece of Python script which will serve the purpose.
- A newly created .py utility file shall be committed into SVN in the path: /<Verification-Branch>/utilities.
- The utility file shall then be subject to peer review.

Structure of each function:

```
def function_Name(XX="", YY="", ZZ=""):
  """
  --------------------------------------------------------------------------------
  Description: This function will ....
  Parameters:
    XX - xxxx xxx xxx
    YY - xxxx xxx xxx
    ZZ - xxxx xxx xxx
  Format: utility_Name.function_Name(XX,YY,ZZ)
  Output: True if xxxx
          False xxxx
  --------------------------------------------------------------------------------
  """
  xxxxxxxxx
  xxxxxxxxx
  xxxxxxxxx
  return xxxx
```

3.2.1.3.1 Widget Mapping

The Doors attribute: "~Widget Mapping" contains Widget Names which match values in the .tdf file that is used to define Datalink A661 pages. We take these values and place them in the file: <Verification-Branch>/utilities/util_Widget_Mapping.py, we only update this file (rather than multiple test procedures) if developers change these widget names.

util_Widget_Mapping.py has the format:

```
. . . . Common elements
#----FUNCTIONS OF REQUEST FUNCTION----------
# ATN FUNCTIONS
PAGE_NAME_ATN_FUNCTION      = "MenuPage_aaATNPage_Name"
# FANS FUNCTIONS
PAGE_NAME_FANS_FUNCTION     = "MenuPage_aaFANSPage_Name"

REQUESTS = {'PAGE_NAME' :{'ATN' : PAGE_NAME_ATN_FUNCTION,
                    'FANS': PAGE_NAME_FANS_FUNCTION}
        }
#---------------------- PAGE NAME----------
#
# TYPENAME WIDGETS
PAGE_NAME_TYPENAME_NAME = "Tdf_Name"
#---------------------- END OF FILE ----------
```

All Tdf names should be validated against the current .tdf file. The .tdf is located:

<Verification-Branch>/vista_sim/Apps/FDSA/Data/definition_files/DLCA6500_DMT.tdf

After making the above change to REQUESTS, the util function gotoPage should work properly to go to your page. For Example:

```
# Got to the VOICE REQUEST FANS Page
util_Test_Support.gotoPage ( logger, VL, "VOICE","FANS")
```

3.2.1.3.2 JSON File with widgets properties:

As the DLCA is an interactive software and UI will have multiple widgets in various types and states, We are in need of consolidated information of each and every particular widget that DLCA supports for the following purposes:

- To identify the location of the widget for the cursor to navigate to that particular location
- To identify the Widget properties like Widget ID, Widget Type, Default Value
- To identify the valid Run-time Modifiable Parameters (RMPs) for a particular widget.

All of this information is consolidated and saved in a JSON format file, Which the parser Libraries(utilities\libs) use to obtain the above mentioned information during the run-time or while

executing the script. Extracting information from JSON file is very quick, Hence the JSON Format is chosen to plot the properties of widgets.

A sample snippet of the JSON file is shown below:

NOTE: The JSON file varies for each and every program as the widgets and its respective properties vary for each program.

Currently, We have 3 JSON files for 3 different programs

- EDS_data.json (for CL604 Environment)
- IPS_data.json (for A220 Environment)
- IPS_dual_data.json (for M170 Environment)
- GS_data.json (for GS Environment)

3.2.1.4 Commenting

- Maximum Line Length: Limit all comment lines to a maximum of 80 characters.
- Comments should be in mixed case, with project acronyms in uppercase.
- Tests shall not have commented–out Python code unless approved by the Lead Verification Engineer.
- Comments associated with every command should always be above the command.
- Adequate comments should be provided against each statement / group of statements.
- Do not reference a Clarification Request number in the test procedure / test case.

3.2.1.5 Test Artifacts configuration guidelines

- For the first time a test procedure is committed into SVN, give the comment as "JIRA WP (if applicable), initial test development".
- While committing initially add the svn:lock property with value = '*' and svn:keywords property with value = "Revision Author Date HeadURL Id"
- For updating files for WP verification, provide the WP ID and brief description. For example

"DLSS-0000XXXX:TC_CPDLC_HMI_002-02 is updated for logon functionality"

- For updating the files for PREP review comments, provide the PREP Review ID and the finding numbers for which the updating is performed. For example:

  "PREP FANSATN-00000XX, findings 1 and 2"

- Do not refer to any of the below while committing the test artifacts.

  a. Person names

  b. RCI / IDC / HCL

  c. Clarification numbers

3.2.2 Software Test Procedure (Alternate Verification Methods):

The Software Test Procedure documents for the alternate verification methods (i.e. host-based testing, unit-testing, and code inspection) serve the same purpose and follow the same format as target-based testing Software Test Procedure scripts. For Software Test Procedure with the alternate verification method, there are special considerations.

3.2.2.1 Code Inspection

For code inspection tests in the Software Test Procedure scripts, steps are needed to get the correct source code file. These steps need to include the archive path and file name. The correct revision of the source code file is needed; it may not always be the latest revision. For each build a Subversion revision tag is assigned to each applicable revision of all the source code files for that build. To view the tag and associated file revisions, do a Subversion show log command on the file. Capture this log output to a file to become a part of the actual results. Open the appropriate Subversion revision of the file to be tested. A step is needed to identify the expected results and what specifically the verifier is to confirm in the source code file (i.e. source code lines in a given order, variable containing certain values, etc.). Based on the expected results the verifier must be guided to arrive at actual results.

Entry Description of test procedure for Code Inspection:

Step 1: In this section Test procedure ID needs to be mentioned.

Step 2: Test Procedure Purpose-> In this section purpose of writing a Test procedure for verifying requirements which are marked as Code inspection / analysis should be written and Type of testing should be identified.

Step 3: DLCA source code file path needs to be mention

Step 4: Test case TC_XXXX_YYYY_### -##:

- Requirement(s) being verified: This section explains about the requirement which is being verified in the test procedure.
- Purpose: Purpose of the requirements being tested in that respective script need to be provided.
- Expected Results: Expected Results of the requirements being tested in that respective script need to be provided.
- Reason for Code Analysis: Reason for performing code inspection or analysis needs to be identified under this section.
- Log Build information like CPN and Build ID need to be logged.

Example of Reason for Code Analysis

1.3.2 Reason for Code Analysis:

( ) Existing test procedures are incomplete. This analysis provides full coverage of the requirement.

(X) Existing test procedures provide no more than partial coverage of the requirement. CI provides supplemental code coverage.

( ) Other, Reason:

- Verification by Analysis: This section explains the part of the code which will verify the requirement through analysis.

3.2.3 Codec Testing

The DLCA delivers a separate Message Library package for the Codec (Code that performs the encoding and decoding of the messages).

The DLCA message library is separately tested using the Message Library Tester tool.

The Message Library Tester tool, also referred to as Codec tool, includes three components – Message tester, Testsupport, Osslibrary.

Message Tester - This module parses the test cases.

OSSLibrary - The Standard library available for the Codec.

Test Support – The wrapper classes + DLCA message library.

The test inputs are passed to message tester, which parses and sends to the osslibrary and the testsupport.

The output results are compared against the expected results given in the test case and/or against the oss generated test results.

The test cases/Procedures are written in normal text documents and saved with an extension of .txt .The results are logged in .csv file with the name same as the name of the testscript.

The .CSV file may contain the below fields depending on the type of the test

CPN Number

TestCase Number

OssPDU Hex

OssMessage Hex

Actual PDU Hex

Actual MSG Hex

Expected Text

Actual Text

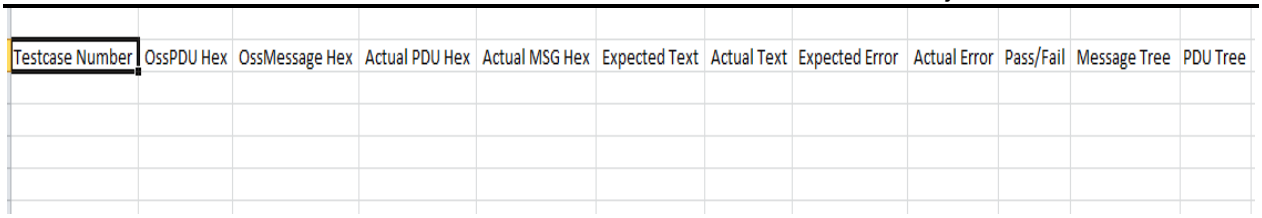
Expected Error

Actual Error

Pass/Fail

Message Tree

PDU Tree



Testcase Number	OssPDU Hex	OssMessage Hex	Actual PDU Hex	Actual MSG Hex	Expected Text	Actual Text	Expected Error	Actual Error	Pass/Fail	Message Tree	PDU Tree

The results file also logs the CPN of the DLCA Message Library on which the test is executed.

3.2.3.1 Naming convention

For ATN CPDLC cases, the Message Library tester tool expects the Keyword ATN in the Testscript name.

So the testscripts are named as ATNXXXXYYY.txt where XXXX can be any of the below:

Downlink

Uplink

InvalidUplink

InvalidDownlink

Pdu

InvalidPdu

YYYY -> any name used to identify the functionality being tested.

For CM, FANS, DM, the test scripts are named to identify the functionality being tested.

Eg. TP_FANS_CODEC_UM0, TP_FANS_CODEC_DM_POS etc.

3.2.3.2 Sample Test Case Templates

For ATN: This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/Codec_Test_Case_Example_ATN.txt

For FANS: This test case template is located: https://asvn/csdlnkver-dlca-a661/documents/Training/Sample_Test_Case_A661/Codec_Test_Case_Example_FANS.txt

Pass/Fail criteria for codec test cases

Note: Below is the logic used to determine the PASS/FAIL criteria

For CPDLCUplink messages, CM groundmessages and CM groundPDUs with messages:

TEST PASS ==if (Expected Error = Actual Error) then if ((Actual Error == 'success') && (Expected Text = Actual Text))

For CPDLCDownlink messages

TEST PASS == (Expected Error = Actual Error) &&

if ((Actual Error == 'success') && (Expected Text = Actual Text) && (OSS Msg Hex = Act Msg Hex) && (OSS PDU Hex = Act PDU Hex))

For DM PDU messages

TEST PASS == (Expected Error = Actual Error) &&

if ((Actual Error == 'success') && (OSS PDU Hex = Act PDU Hex))

For CM Aircraft messages

TEST PASS == (Expected Error = Actual Error) &&

if ((Actual Error == 'success') && (Expected Text = Actual Text) && (OSS PDU Hex = Act PDU Hex))

For AircraftPDUs without any Downlink message:

check PDU tree for AircraftPDUs

TEST PASS == (Expected Error = Actual Error) &&

if ((Actual Error == 'success') && (Expected Text = Actual Text) && (OSS PDU Hex = Act PDU Hex))

For GroundPdus without any uplink message

check PDU tree for groundpdus

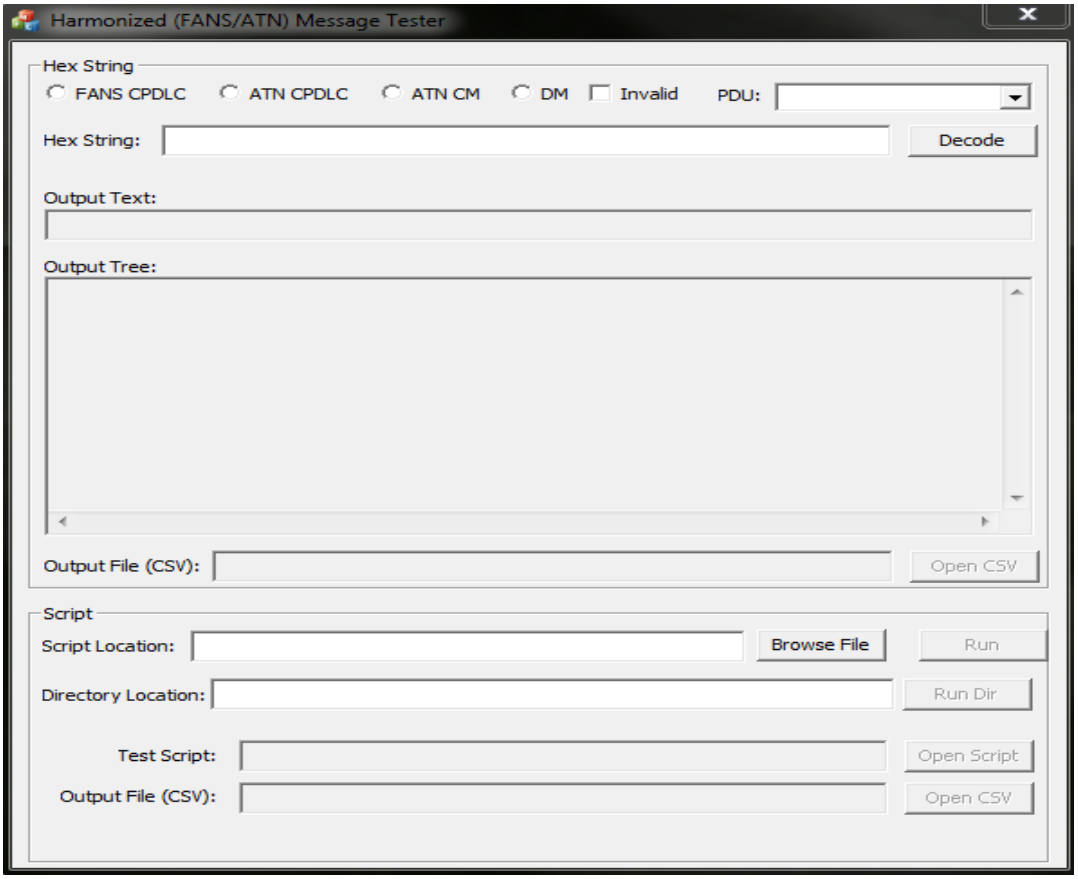
TEST PASS == (Expected Error = Actual Error) &&

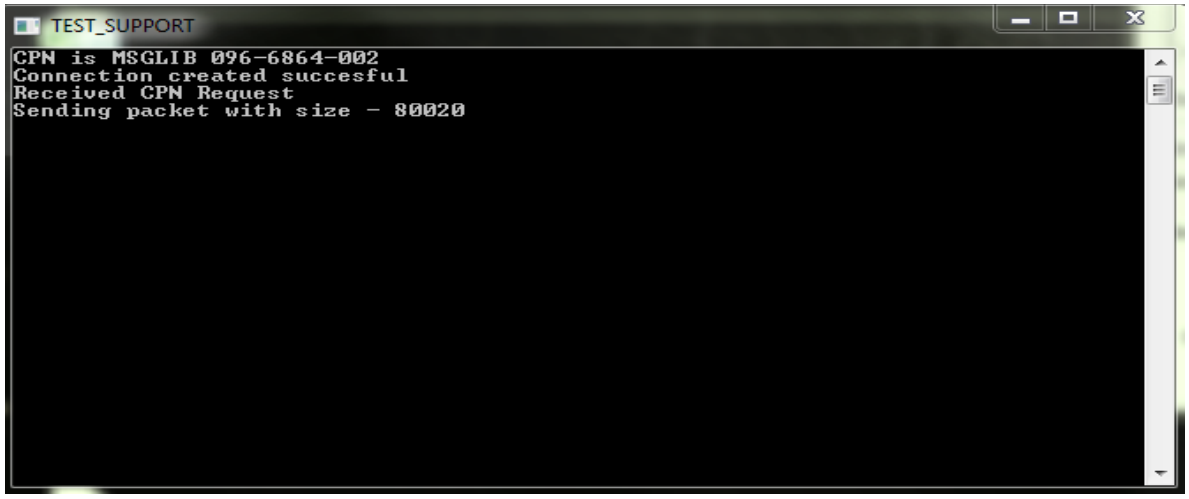
if ((Actual Error == 'success') && (Expected Text = Actual Text))

3.2.3.3 Sample Test Execution

(i) Go to vista_sim folder present in SVN path https://asvn/csdlnkver-dlca-a661/xxxxx/

(ii) Run CODEC_Host and the following screens will be generated as shown in the figures.





(iii) Click on Browse file in message tester tool and select the CODEC file you want to execute present in the path <Verification-Branch>/test_procedures/CODEC .

(iv)After the execution is complete a CSV file is generated in the same folder where the CODEC scripts are present. The format of the CSV file is shown in the figure:

CPN :	MSGLIB 096-6864-002										
Testcase Number	OssPDU Hex	OssMessage Hex	Actual PDU Hex	Actual MSG Hex	Expected Text	Actual Text	Expected Error	Actual Error	Pass/Fail	Message Tree	PDU Tree

3.3 Test Case Summary Table

The Test Case Summary (TCS) is an Excel table (*.xlsm), which provides a summary of executions of each test case covered, when a test procedure is run both with an functional (uninstrumented) software build and structural (instrumented) software build.

The TCS is required to be updated if it is the last expected run of the test procedure and also each time a test procedure is rerun. When rerunning tests, the latest test procedure and associated files will be referenced in the TCS.

Requirements based test procedures will be performed twice: once without instrumentation and again with instrumentation. The same results should be realized. If there is a difference in results, then a JIRA WP must be opened.

3.3.1 Entry Descriptions

The purpose of the sheet is to track results from both types of builds, and also to identify whether or not the test procedure with its test cases is executed for structural coverage data collection. The build must be instrumented to collect structural coverage data.

The TCS should reflect only the last results used for the run-for-score verification effort. History may be obtained by going back to previous SVN revisions of the file.

Since we support multiple programs and environments, We maintain separate tabs for IPS Productline, EDS Productline, EDS ALL, IPS ALL, NONCONFI_INBOX, M204, A220 tabs

There is a separate sheet maintained for CODEC tests located in the same path where the original TCS is present.

TCS Entry Descriptions:

Test Script/ Log file Name:

The Test Procedure associated with the TCS table. Test Procedure name will be mentioned.

Test Case:

This column indicates the number of test cases associated with each test procedures.

S/W Build:

The software build the Test Case was executed against. The s/w build will be an incremental build, a prerelease or release build AND either instrumented or uninstrumented.

Host Result Status (Pass/Fail):

Pass means the actual results meet the criteria established by the expected results identified in the Test Case. Fail means the actual results do not meet the criteria established by the expected results identified in the Test Case in the HOST environment.

SCA Result Status (Pass/Fail):

Pass means the actual results meet the criteria established by the expected results identified in the Test Case. Fail means the actual results do not meet the criteria established by the expected results identified in the Test Case when run on sca environment.

Target Result Status (Pass/Fail):

Pass means the actual results meet the criteria established by the expected results identified in the Test Case. Fail means the actual results do not meet the criteria established by the expected results identified in the Test Case when run on target environment.

WP:

For all Test Cases that fail, a JIRA (WP) number must be listed. The WP will track the failure and will be referenced in the Software Accomplishment Summary. This WP should not be in the closed state.

WP/Note:

TCS Entry Descriptions:

Information can be provided concerning why a test case may have failed, whether the test is automated or manual, if an SQE witnesses the execution of a Test Case, etc.

Test Environment:

Different Test Environments like Default, Dual DLCA, Dual AFD, Dual DLCA_AFD2, Inbox need to set according to the respective script

Test Method:

Different Test Methods like Automated, Inspection, Manual, Analysis need to set according to the method used in respective script.

No.of csv files per TP:

During Execution of automated test scripts, The TPs will create and save few csv files with A661 trace which will be used by the utilities. This cell will have "N/A" when there are no csv files or the integer number of the csv files that will be generated for that respective script.

Remarks:

This column indicates any extra information required to validate the failures for example if any test case is failing due to host environment issues only then it is mentioned here. This may also indicate any specific information required to execute the script etc. This column may also indicate if the script is required only for HOST/TARGET etc. This column also refers if the test procedure requires any custom build or not.

Error WP:

In case of test cases where there are no failures but errors are there then this column indicates the WP for the errors. In cases where there are failures and errors then the test case is considered as fail and there is one WP to identify these issues which is mentioned in **CR** column and errors CR column is left blank.

P/A:

This column is meant for Product Applicability. "All" stands for both EDS and IPS environments.

"EDS ALL" stands for EDS only environment. "IPS ALL" stands for IPS only environment. "M204" stands for M204 program. "A220" stands for A220 program.

3.3.2 TCS Template for all tests and CODEC tests

Table 2: Template of TCS

	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Test Case	Build	Host ResultStatu	SCA ResultStatu	Target Result Sta	WP	WP/Note	Test Environme	Test Method	. of .csv files(per	Remarks	Error WP	P/A
2	TC_ACF_001-01	5.0.6	PASS	PASS				Default	Automated	N/A			All
3	TC_ACF_001-02	5.0.6	PASS	PASS				Default	Automated	N/A			All
4	TC_ACF_001-03	5.0.6	PASS	PASS				Default	Automated	N/A			All
5	TC_ACF_001-04	5.0.6	PASS	PASS				Default	Automated	N/A			All
6	TC_ACF_002-01	5.0.5	PASS	PASS	PASS			Default	Automated	N/A			All
7	TC_ACF_002-02	5.0.5	PASS	PASS	PASS			Default	Automated	N/A			All
8	TC_ACF_002-03	5.0.5	PASS	PASS	PASS			Default	Automated	N/A			All
9	TC_ACF_002-04	5.0.5	PASS	PASS	PASS			Default	Automated	N/A			All
10	TC_ACF_003-01	4.0.2	PASS		PASS			Default	Manual	N/A			All
11	TC_ACF_004-01	4.0.2	PASS		PASS			Default	Automated	N/A			All
12	TC_ACTIVE_STANDBY_001-01	5.0.2a	PASS		PASS			Dual DLCA	Automated	N/A	HOST in executed on 4.0.2a IPS_TARGET in execu		All

Table 3: Template of TCS for CODEC tests

Test Case Path :					
Functional Run Result (Target) Path:					
Structural Run Result (Host) Path:					
Test Scripts	S/W Build	Functional Run Result (Target)	Structural Run Result (Host)	CR_ID(s)	CR_Note

3.4 Structural Coverage Analysis

The process steps involved for Structural Coverage Analysis (SCA) involve instrumenting the software, testing the instrumented software on the host, analyzing the results from the testing, recording the results from the analysis, and creating a summary report of the SCA results.

Note: SDK version has been upgraded from Vista_MC_4.5.0 to Vista_MC_5.0.0

3.4.1 Instrumenting Software

VectorCAST tool is used to instrument the DLCA software with DO178B Level C instrumentation.

Note: Refer DLCA-Developers Guide for Instrumenting Software in Section 19

https://asvn/dlnk-dlca/branches/Reference/DLCA-DevelopersGuide.docx

3.4.2 Testing with Instrumented Software

Test Procedures will be run on the instrumented and the non-instrumented builds provided from the development team.

3.4.3 Analyzing Coverage Results

The coverage test results from the instrumented build are imported into the VectorCast Cover tool. Then a coverage report is created for each instrumented file which combines the coverage test data with the source code. The covered lines of code are annotated by the tool with a symbol '*'. The noncovered lines are not annotated and require further analysis as to why the code was not covered. The analysis of the uncovered lines of code is documented in the coverage report for each file. The coverage reports for each file are placed under configuration management control.

The process steps to accomplish the above are as follows:

- The test results from the instrumented run are combined into one .dat file. The combined .dat file has the duplicate test points removed so that only one test point exists for each line of covered code.

- The combined .dat file is imported into VectorCast Cover tool using the "Add test result…" icon.

- To generate the coverage report, select all files and then right click and select "Export to folder". Select the "Coverage" and "Basis Paths" options.

- The filename *_coverage.txt file has the covered lines of code annotated. The lines that are not covered have coverage analysis completed on why the lines are not covered. These comments are documented in the "*_coverage.txt" file and placed under configuration control. See section *SCA Reason Codes for Uncovered* for the reason of uncovered code.

- Each *_coverage.txt file has a predefined template header added at the top of the file with information that includes the build ID, file version analyzed, reason for code analysis, result of the code analysis including any JIRA WP that were written. Each section of code that is not covered will have an analysis template filled out.

- When completed, the *_coverage.txt file is added to development configuration management control.

For more details, refer section 4.7 and refer the document for how to perform detailed structural coverage analysis in

https://asvn/csdlnkver-dlca-a661/branches/4.0.0_Ver/tools/SCA Tools/VCAST2Annotated extractor/Read-Me.docx

3.4.3.1 SCA Reason Codes for Uncovered

Reasons that the Data Link Coverage Analysis results files show uncovered structures are listed below. All uncovered structures are analyzed, and notations are made in the files with a letter code that matches these reasons.

Refer section 7.3.2.2 Structural Coverage Analysis Method of the Software Development Plan [7] for more details.

4 Verification Artifact Naming Conventions

The verification artifact naming conventions are provided to consistently and uniquely identify verification artifacts.

HMI Test Cases/Procedures will be named TP_CPDLC_HMI_XXX.py

HMI Test Cases/Procedures will be named TP_HMIX.py for Program specific HMI Test procedures where X denotes prefix for Program.

Example: TP_HMIK_XXX.py for KC390, TP_HMIM_XXX.py for MRJ

TP Naming Conventions for all projects will be same but the prefixes will be changed according to the requirement section in respective SRS

4.1 Functional and Sub Functional Area Codes

Naming conventions use the functional and sub functional area codes identified for the DLCA. The following table contains the codes.

Table 4: List of Functional and sub functional area codes

Functional Areas	
Code	**Description**
ACF	ACARS Convergence Function
ACTIVE_STANDBY	Active Standby
ADS	Automatic Dependent Surveillance
AFN	ATS Facilities Notification
ALERTING	Alerting Capabilities
APP_HOST	Application Hosting
ATN_CPDLC	Core ATN functionality
BCAST	Broadcast
CM	Context Management
Code_Inspection	Code Inspection
CODEC	ASN.1 Codec Decoding/Encoding
CPDLC_HMI	HMI for CPDLC
CPN	Collins Part Number
CSS	Common System Services
DMGR	Data Manager
DSI	Dialog Service Interface
FANS_CPDLC	Core Future Air Navigation System functionality
INBOX	Inbox and QAK
MEMORY	Memory management
NONCONFIG_INBOX	Nonconfig Inbox scripts
SYSCLOCK	System Clock/Time Management
ATNE	ATN Functionality EDS Specific
HMIE	HMI Functionality EDS Specific
HMIM	HMI Functionality MRJ Specific
CSSE	CSS Functionality EDS Specific
TIMING	Timing related scripts.
DISPLAY_FORMAT	DISPLAY FORMAT common Test Procedures. DISPLAY FORMAT is applicable only to IPS environment as for EDS we don't own the HMI.

Functional Areas	
Code	**Description**
HMI(X)	HMI Test Procedures specific to Program (X)
SYSTEM_TEST	System Testing

4.2 Test Procedure Identifiers

Each individual Test Procedure within the DOORS Software Test Procedures is assigned a Test Procedure ID according to the naming convention below. The Test Procedure ID will be used for identifying the Test Procedure in the Test Case Summary table. The Test Procedure ID is dependent on the alphabetical order of the Test Procedure ID in the Software Test Procedures document.

Test Procedure ID	TP_ XXXX_###.py 'TP_' is a constant which stands for Test Procedure 'XXXX' – functional area '_' a constant '###' – A unique number to differentiate this Test Procedure amongst the others in this Functional area (start with 001). The applicable Software Test Procedures document should be consulted for the next available last three numbers, ###. .py – a constant extension for Python script files Example: TP_CPDLC_HMI_002.py This Software Test Procedure is for HMI functional area. TP_DMGR_001.py This Software Test Procedure is for DMGR functional area. TP_ADS_001.py This Software Test Procedure is for ADS functional area. Path: /test_procedures/CPDLC_HMI/TP_CPDLC_HMI_001.py /test_procedures/DMGR/TP_DMGR_001.py /test_procedures/ADS/TP_ADS_001.py

TP Structure	Test Procedure Revision Information Test Case Header Test Case 1 Start of Test Procedure for Test Case 1 End of Test Procedure for Test Case 1 Test Case 2 Start of Test Procedure for Test Case 2 End of Test Procedure for Test Case 2

4.3 Test Case Structure

Each Test Procedure (*.py file) will contain multiple Test Cases. Each Test Case will be identified by its Test Case ID according to the naming convention below. The Test Case name would be incremented with -01, -02 along with the Test Procedure name except that it starts with a TC_ rather than TP_.

Rules while writing Test Cases:

1. When a requirement is tested partially, it is mandatory to specify it in the test case header along with the requirement as below:

```
#+================================================================================+
#|================================================================================|
#|
|
#|TEST CASE   : TC_CPDLC_HMI_151-01                                               |
#|                                                                                |
#|================================================================================|
#+================================================================================+

#|REQUIREMENTS: HMIC3667, HMIC7090, HMIC7259, HMIC11677, HMIC13168 [Partial]   |
#|PURPOSE     : To Verify that                                                    |
```

2. The Purpose should be specific on what is being tested (and not the whole requirement)

```
#|REQUIREMENTS: HMIC26343 [Partial]                                              |
```

Bad Purpose:

```
#|PURPOSE     : To Verify that                                                    |
#|              The SEND FUNCTION will be {Enabled} on the CPDLC pages when the |
#|              following conditions are met:                                     |
#|              - The VERIFY MSG AREA has downlink message text                   |
#|              - The generated message text has been accessed entirely by       |
#|                scrolling since the page entry (See Note 1)                    |
#|              - The CPDLC application has a CDA Value on the LOGON/STATUS page|
#|              - The Inactive Advisory "NO COMM" is disabled                     |
#|              - The Inactive Advisory "NO RIU CONN" is disabled                 |
#|             - No FANS message is waiting to receive a Network Acknowledgement|
#|               from MU or the FANS message has timed out waiting for Network  |
#|               Acknowledgement from MU.                                        |
#|              - Message repository should not have reached its maximum         |
#|                capacity                                                       |
#|              - When responding to a FANS uplink or generating a FANS pilot    |
#|                initiated downlink (FANS Only)                                 |
#|              - When responding to a ATN uplink message that does not include |
#|                an ATN End Indication or generating a ATN pilot initiated      |
#|                downlink (ATN Only)                                            |
```

```
#|             - When the ATN AVAILABILITY LABEL is {Invisible} (ATN Only)     |
Good Purpose:
#|PURPOSE      : |

#|              [HMIC22076] The CLEAR Function should be disabled when any    |
#|                          editable field does not contains an entry         |
                 |
```

3. A requirement can only be marked [Partial] if there are OR conditions in the requirement and not AND

4. Minimize the number of requirements tested in a single test case. (Advised not to have more than 5 OIDs)

Each Test Case will be composed of:

- Test Case ID: TC_XXXX_####-##
- Requirements list: HMI-XXXX, HMI-XXXX, HMI-XXXX
- Purpose: To Verify XXXXX XXXXXXX
- Test Type: Normal / Boundary / Out of Range Testing / Equivalence class
- Condition: XXXXXXXXX XXXXXXXXX
- INPUT: XXXXXXX
- Expected Results: Verify XXXXXX XXX

Test Case ID	TC_ XXXX_### - ## 'TC_' – a constant which stands for Test Case 'XXXX' – which stands for functional area '_' a constant '###' – A unique number to differentiate this Test Cases amongst the others in this Functional area (start with 001). '-'a constant '##' – A unique number to differentiate this Case amongst the others in this Test Procedure (start with 01) Example: TC _CPDLC_HMI_001-01 (Its Test Procedure name is TP_CPDLC_HMI_001)

4.4 Test Procedure Standards

- When writing test procedures, try to create versions that could run on multiple platforms without modification wherever possible. Doing this may require specific experience or guidance from senior test engineers.
- Use logger.infoWrap before every testWrap to indicate in the log file what is being tested. For example:

```
logger.infoWrap('Verify downlink message for REQUEST VMC DESCENT DM69')
logger.testWrap('util_Test_Support.findWidgetEvent(trace=True,widget=VE
RIFY_MSG_LABEL1,parameter=[WIDGET_TEXT],instance="L_AFD_DLCA")==[verify
Msgs[0]]')
```

- Use blockParser and resulting string array to verify message text that may or may not span multiple lines. For example:

```
# Parse the initiated downlink message, store in string array.
```

```
    Verify_Msgs = util_Test_Support.blockParser('REQUEST VMC DESCENT, DUE
    TO AIRCRAFT PERFORMANCE')
    # Ensuring length of message = 2 lines.
    Length = len(verify_Msgs)
    logger.testWrap("length == 2")
    logger.infoWrap('Verify downlink message REQUEST VMC DESCENT, DUE TO
    AIRCRAFT PERFORMANCE. DM69, DM66')
    logger.testWrap('util_Test_Support.findWidgetEvent(trace=True,widget=VE
    RIFY_MSG_LABEL1,parameter=[WIDGET_TEXT],instance="L_AFD_DLCA")==[verify
    Msgs[0]]')

logger.testWrap('util_Test_Support.findWidgetEvent(trace=False,widget=VERI
FY_MSG_LABEL2,parameter=[WIDGET_TEXT],instance="L_AFD_DLCA")==[verifyMsgs[
1]]')
```

4.5 Utilities

The Utilities are Python programs that are designed to serve as reusable "building blocks" for constructing test files, by encapsulating functions that are commonly included in more than one verification test.

Purpose and Use: Purpose and Use of using Utility is to save time and to optimize the script by avoiding rewriting an existing segment of python in scripts wherever is required.

Utilities	util_XXXX.py XXXX – Specific name of the function the py file performs when it is called. .py – a constant extension for python script files Syntax for using utilities: import util_XXXX execfile(os.environ["DLCA_UTILS"]+"\\util_XXXX.py") Path : /utilities

4.6 Test Procedure Actual Result Filenames

Test Procedure actual result files may be automatically or manually created as stated in the Test Procedure document. The actual result filenames are used for uniquely identifying the files and for referencing the filenames in the Test Procedure document, as well as the Test Case Summary table.

Actual Result Filenames	Same as Test procedure name with the extension as .LOG instead of .py

4.7 Structural Coverage Results Filenames

Each Test Procedure will have associated structural coverage .dat and .log files, which will be identified according to the naming convention below.

Structural Coverage .DAT folder	/sc_results

Structural Coverage Results Filenames	TP_XXXXX_YYY.LOG TP_XXXXX_YYY.DAT 'XXXXX' – the functional area of specific Test Procedure 'YYY' – Test Procedure ID .LOG – contains logged information about all the test cases covered in that test procedures – pass or fail details, etc. .DAT – contains information regarding the lines of code covered while executing that particular test procedure Example: TP_FANS_CPDLC_005.LOG, TP_FANS_CPDLC_005.DAT, This file contains the structural coverage results associated with the Test Procedure ID for functional area FANS_CPDLC, number 005. TP_FANS_CPDLC_005.LOG – SCA Log file for FANS_CPDLC_005 file TP_FANS_CPDLC_005.DAT – SCA .dat file for FANS_CPDLC_005 file

The .DAT files are placed in the " http://asvn/csdlnkver-dlca-a661/sca_analysis/XXXX ",

where 'xxxx' refers to the build from which the .DAT files were obtained, for example

XXXX =Build_5.0.10

The Annotated results are placed in " http://asvn/csdlnkver-dlca-a661/sca_analysis/Annotated_results ".

These capture the uncovered SLOC(s) from respective source code.

- For each .cpp and .h, the corresponding coverage analysis result files are generated containing

only the coverage gaps and their corresponding analysis below that coverage gap.

- The tool used to extract these coverage gaps from the VectorCAST coverage report files is placed in

"<Verification-Branch>/tools/SCA Tools/ DLCA6500_Gen_VCast2Annotated.py ".

- Refer " https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/tools/SCA Tools/VCAST2Annotated extractor/Read-Me.docx" document for more details about the tool, where x.x.x_Ver is 4.0.0_Ver

4.8 Software Build Identifiers

Each software build is assigned a unique identifier to distinguish the build from all others according to the naming convention below. The Software Build ID will be used for identifying the build that is loaded in the

IPC Cabinet. The Software Build ID is used in the Test Case Summary table.

Software Build ID	Software Build ID: Build# X.XX.X, where X = 0-9 ex - 1.13.2 or 2.0.1 Loadset RL_EModXX_A220_VX.X.XX_XXXXX_YYYY_MM_DD 'XXXXX' – numeric characters to uniquely identify the software build Example: RL_EMod02_A220_V5.0.10_25732_2022_12_12

4.8.1 Available Software Builds

This section is used to keep track of software build IDs used for run-for-score verification. The latest software build for a project should be used.

SVN Build
5.7.0

5 Verification Configuration Management

5.1 Subversion (SVN)

5.1.1 Access

Subversion is the configuration management tool used for the DLCA 6500 Verification project. The path for the DLCA 6500 Verification SVN repository is http://asvn/csdlnkver-dlca-a661/branches/5.0.0_Ver (the paths may differ from project to project). In order to gain access to the repository, a user must be included in either the EBSS group called alliance\CS-DLNKVER-DLCA-A661-RO or alliance\ CS-DLNKVER-DLCA-6500-RW. Please see the Data Link Verification Lead, Engineering Project Assistant (EPA), or Engineering Project Specialist (EPS), if access is required. For foreign nationals, update the Access Control Plan (ACP) to include DLCA 6500 Verification SVN repository prior to providing access.

In order to log in to the repository, special attention must be paid to what domain the user is logging in from. The repository has been created on the alliance domain. If a user is logging in from the ccanet domain (most users on the Cedar Rapids-based data link team), the user must specify so. For example:

Username: ccanet\prrichar

Password: (normal ccanet password)

If a user is logging in from the RCINNET domain (most users on the IDC-based data link team), the user must specify so. For example:

Username: RCINNET\mteegava

Password: (normal RCINNET password)

5.1.2 Verification Artifact Locations

The Subversion directory associated with verification is located in the SVN repository path http://asvn/csdlnkver-dlca-a661/branches/5.0.0_Ver. The subdirectories include the following for verification artifacts:

Based on the project needs, some of the subdirectories will also be present in branches.

\actual_results	for result/log files obtained from running test procedures/scripts
\Archive	for old and unused scripts
\Reference	for the purpose of support and Reference
\sc_results	for DAT files obtained from instrumented runs.
\tcs	for Test Case Summary tables
\test_procedures	for Test Procedure executable files and Test Procedure support files
\test_procedures_EDS	for Test Procedure executable files and Test Procedure support files
\test_procedures_KC390	for Test Procedure executable files and Test Procedure support files
\test_procedures_M204	for Test Procedure executable files and Test Procedure support files
\test_procedures_MRJ	for Test Procedure executable files and Test Procedure support files
\test_procedures_Program	for Test Procedure executable files and Test Procedure support files
\tools	for tools
\TRRs	for Test Readiness Reviews
\utilities	for utility files
\ vista_apps	for vista applications
\vista_sim	for DLCA test tools and environment

\ vista_sim_EDS for DLCA test tools and environment for EDS Environment (CL604)\
vista_sim_EDSDual for DLCA test tools and environment for EDS Environment (M204)

\ vista_sim_IPSDual for DLCA test tools and environment for IPS Dual DLCA

\ vista_sim_MRJ for DLCA test tools and environment for MRJ

\vista_sim_GS for DLCA test tools and environment for GS

The Subversion directory associated with SCA report files and annotated results is located in the SVN repository path http://asvn/csdlnkver-dlca-a661/sca_analysis.The subdirectories include the following for verification artifacts:

\Annotated_results	Captures the uncovered SLOC(s) from respective source code
\Build_xxx1	Contains DATs and coverage reports(HTML and text reports) for Build_xxx1
\Build_xxx2	Contains DATs and coverage reports(HTML and text reports) for Build_xxx2

The subdirectories of every Build folder, i.e. http://asvn/csdlnkver-dlca-a661/sca_analysis/Build_xxxx will be,

as follows:

\DAT_Files	.DAT files obtained from instrumented run
\VectorCAST_Results	Contains HTML and Text coverage reports (extracted from VectorCAST tool)

e.g For RFS 3.x the paths are as follows:

DATS path for RFS 3.X is http://asvn/csdlnkver-dlca-a661/sca_analysis

for reports http://asvn/csdlnkver-dlca-a661/sca_analysis

Verification Artifact Naming Conventions & Subversion Directories:
http://asvn/csdlnkver-dlca-a661/branches/5.0.0_Ver/

```
actual_results/
        5.0.3_dry_runs/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        5.0.5_dry_runs/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        Displayformat/
            XXXXXX/TP_DISPLAY_FORMAT_xxx.LOG  (Eg: TP_ DISPLAY_FORMAT
    _001.LOG)

        dry_runs/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        IPS_runs/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        RFS_3.x/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        RFS_5.1/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        RFS_5.2/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        RFS_5.x/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        sca_dry_runs/
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        target_dry_runs
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
        test_runs
            XXXXXX/TP_CPDLC_HMI_xxx.LOG      (Eg: TP_CPDLC_HMI_001.LOG)
```

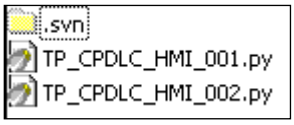
```
tcs/
        tcs_dlca_a661_idc.xlsm
        tcs_dlca_a661_codec.xlsm
        Archive\
        tcs_dlca_a661_idc_Delta_RFS_5.2.xlsm
        tcs_dlca_a661_idc_RFS.xlsm
        tcs_dlca_a661_codec_RFS.xlsx


test_procedures/
        ACF/
        ACTIVE_STANDBY/
        ADS/
        AFN/
        ALERTING/
        APP_HOST/
        ATN_CPDLC/
        BCAST/
        CM/
        CODEC/
        CPDLC_HMI/
        CPN/
        CSS/
        Code_Inspection/
        DMGR/
        DSI/
        FANS_CPDLC/
        INBOX/
        MEMORY/
        NONCONFIG_INBOX/
        SYSCLOCK/
        TIMING/
```

5.1.3 General Commands in SVN

5.1.3.1 Settings:

All files are read only until a lock is obtained on the file to be edited. This is indicated by the grey checkmark



A lock is obtained by right clicking on the files and selecting Get lock in TortoiseSVN. If a lock cannot be obtained because of a wrong version or it is already obtained by someone else, you will be prompted to update your version before obtaining the lock. On this project, locks are required to edit files and stealing locks is not permitted without leadership approval.

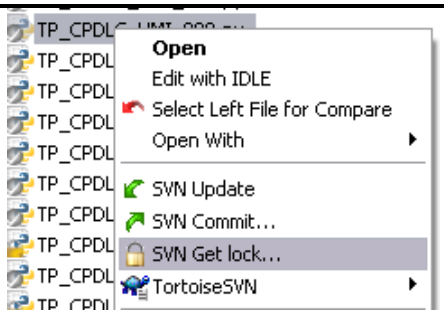


Figure 4: Get lock in TortoiseSVN

5.1.3.2 Update

The working copy of the repository is static. It has to be updated occasionally to get all the update versions. This should be done when reviewing tests and editing files. This will not affect the repository.

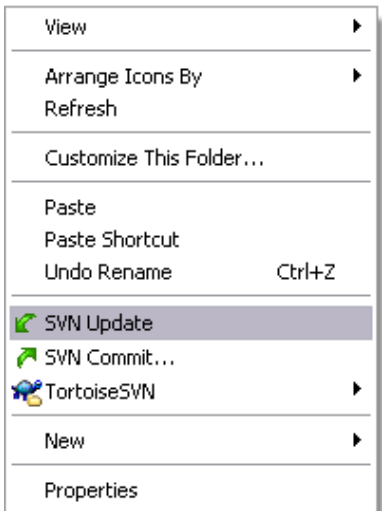


Figure 5: SVN update in TortoiseSVN

5.1.3.3 Commit

After editing the file, it should be committed into SVN at the appropriate path. This will update the repository version and add the updated file to the current Repository. This cannot be undone. The repository can be reverted to a previous configuration but will still result in updating the repository. While committing the file, the comment has to be provided which should be short and specific. If a JIRA WP or Peer Review has prompted the commit, this should be noted in the comment.

SVN commit should be performed only when Test scripts are newly created / updated either for Peer Review comment/ JIRA WP.

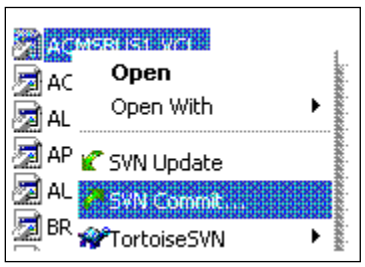


Figure 6: SVN Commit in TortoiseSVN

For committing any script (Test Case, Test Procedure or utility), we need to include JIRA WPs at the top most right side.

Figure: 6.1 JIRA WP while committing files

5.1.3.4 Comparing Versions

This can be done by looking at the log for the individual file and using the Ctrl key to select the versions you would like to view as shown below:

View the log:

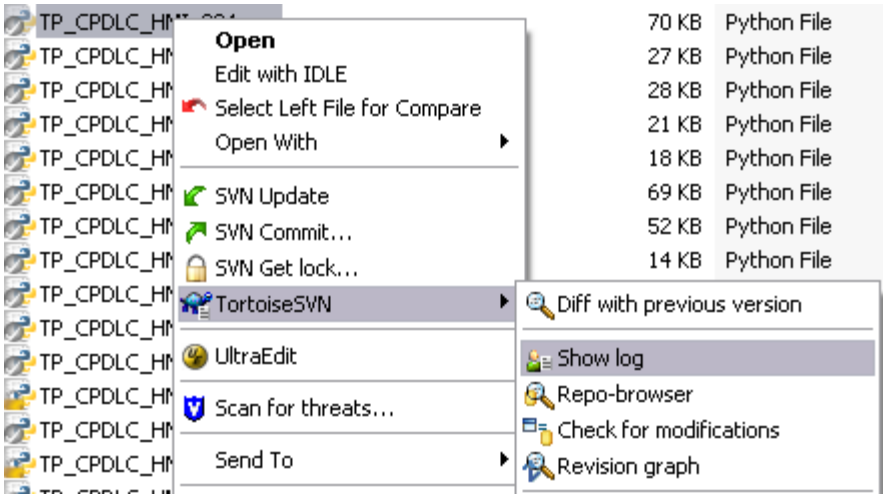


Figure 7: Show Log in TortoiseSVN

Compare the revisions:

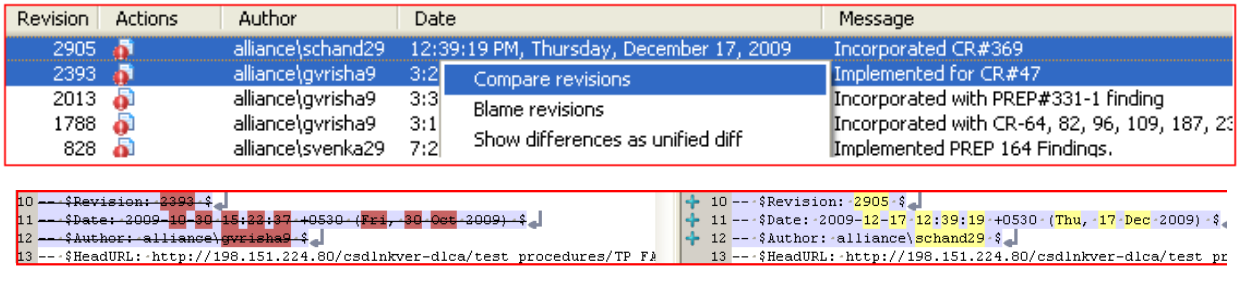


Figure 8: Compare revisions in TortoiseSVN

5.1.3.5 Editing

Once the file is edited, the icon of the file changes to red as shown below:

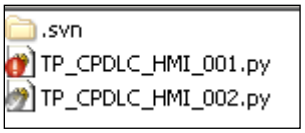


Figure 9: Edited files in TortoiseSVN

Once the icon changes as shown above, the file will likely to be committed (if that was the user's intent) as explained in sec 5.1.3.3 Commit

5.1.3.6 Adding the files

Files can be added by simply copying the file into the folder under Subversion Control

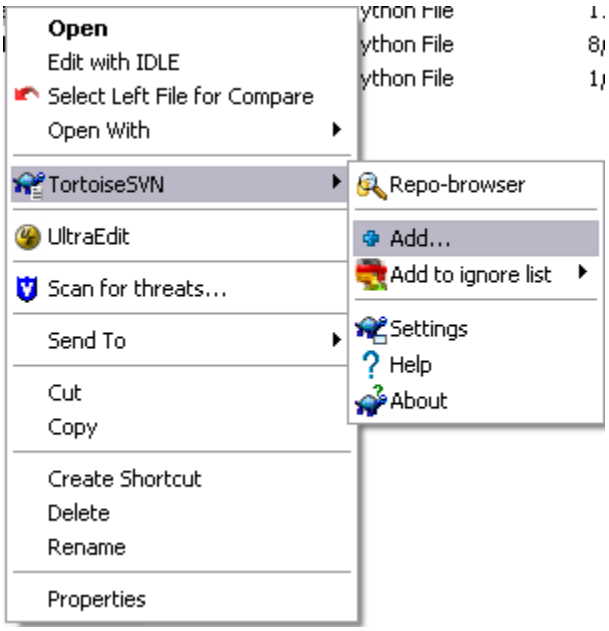


Figure 10: Adding files in TortoiseSVN

Once the files are added as above, the SVN properties for the file should be updated: needs-lock=*, keywords="Revision Author Date HeadURL Id". The file should then be committed as explained in section 5.1.3.3 Commit

Ensure Required SVN properties are added to the new file added before committing it.

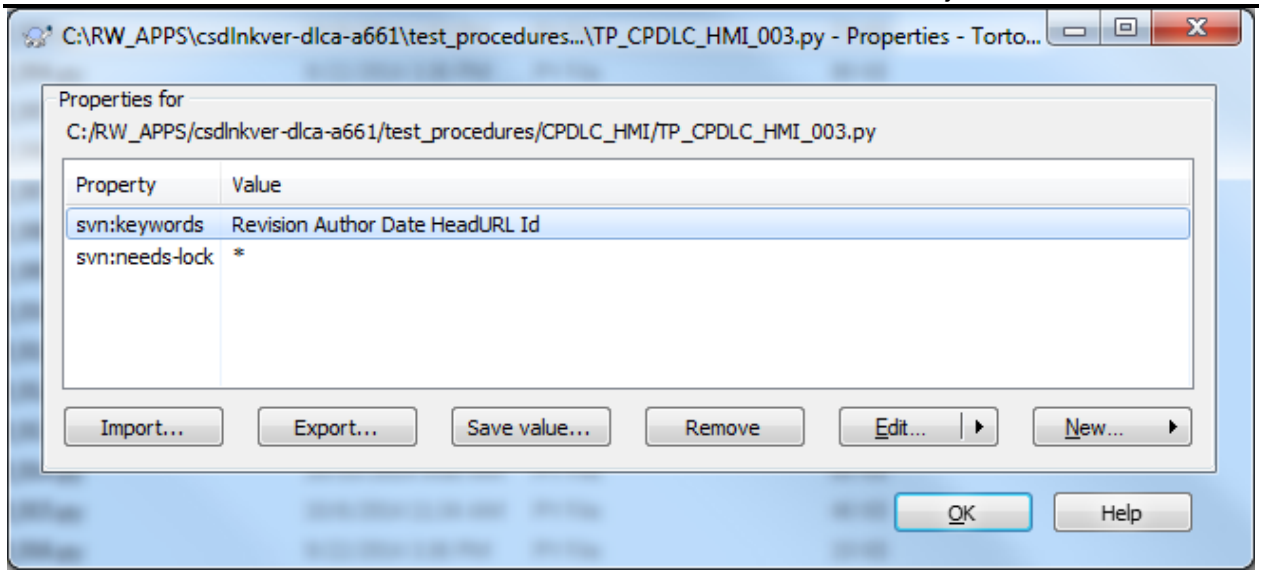


Figure 11: SVN properties

5.1.3.7 Blame

This feature lets you see who edited which lines of a file. Hovering over the line will give information on what was done?

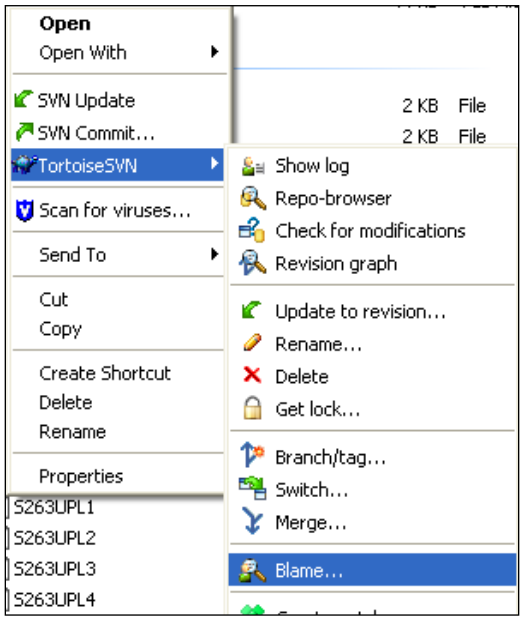


Figure 12: Blaming files in TortoiseSVN

Example:

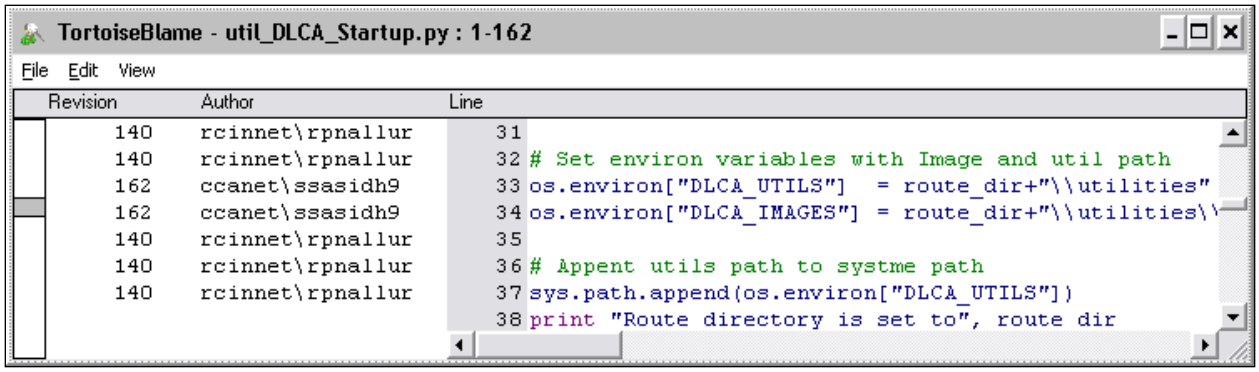


Figure 13: Example for Blaming files in TortoiseSVN

5.1.4 Release of Archived Files

At the end of a project, all test verifications files must be archived and released as a CPCI for certification. These files include test result files, test procedures, test scripts and test support files. These files are stored in SVN during testing and development. The following is a method to copy the test artifacts / results into the SVN release folder to facilitate the release of the artifacts.

5.1.4.1 Copying to the release folder

The project determines the version label to be used for the release. Some typical names are: 815-2453-001(CPCI part number), AOA_Eng_Build_20A, ATN_Eng_Build_7D, etc. Once the project has determined the Version Label Name, follow the next steps in SVN to copy the artifacts into the release folder.

Open the Repo browser with the address with the respective project address

Right Click on the able_builds folder and select "Copy to" as below:

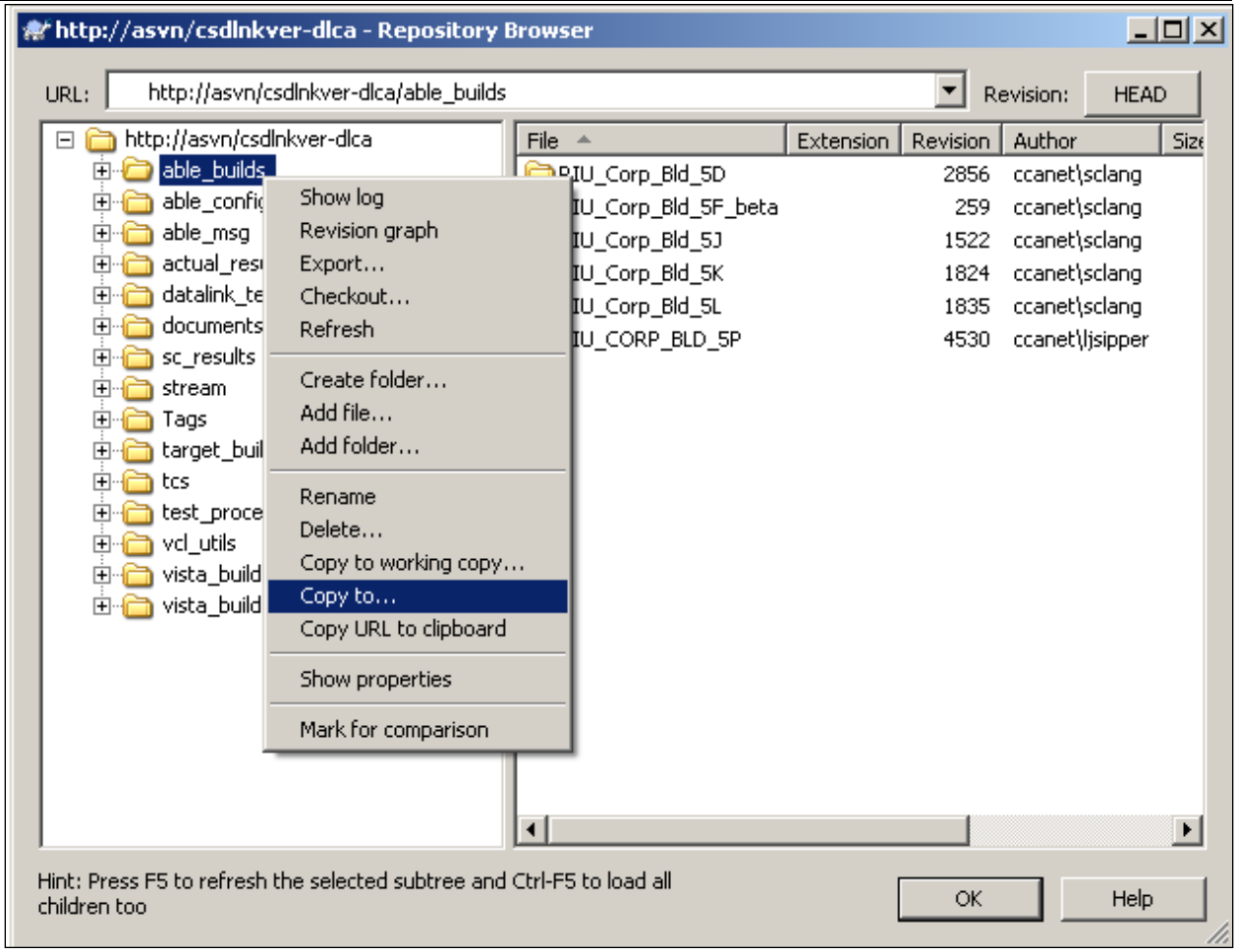


Figure 14: SVN – Copying the files to release folder

Provide the below path and click OK:

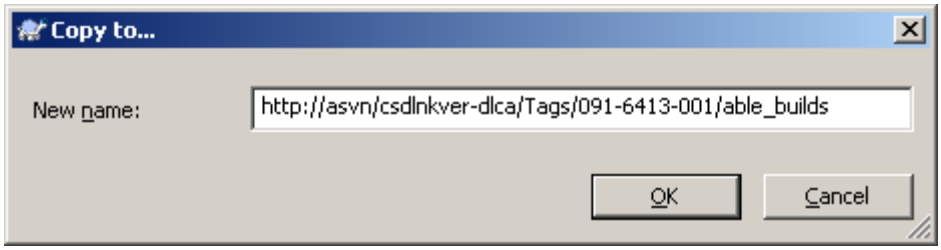


Figure 15: SVN – Copying the files to release folder

Similarly copy the below folders into the Tag folder (/Tags/091-6413-001/):

/actual_results

/sc_results

/tcs

/test_procedures

/utilities

/vista_sim

5.1.4.2 Archiving the Results using an Archive Only CPCI release

Once the artifacts have been copied into http://asvn/xxxxxx-xxxxx-xxx/Tags/XXX-XXXX-XXX/, a copy of this folder needs to be zipped and released. Create a folder C:\Temp\XXX-XXXX-XXX and create 2 folders csdlnkver-dlca-a661. Export all the folders under http://asvn//xxxxxx-xxxxx-xxx /Tags/XXX-XXXX-XXX/ into csdlnkver-dlca-a661, so that all the SVN backup files are removed.

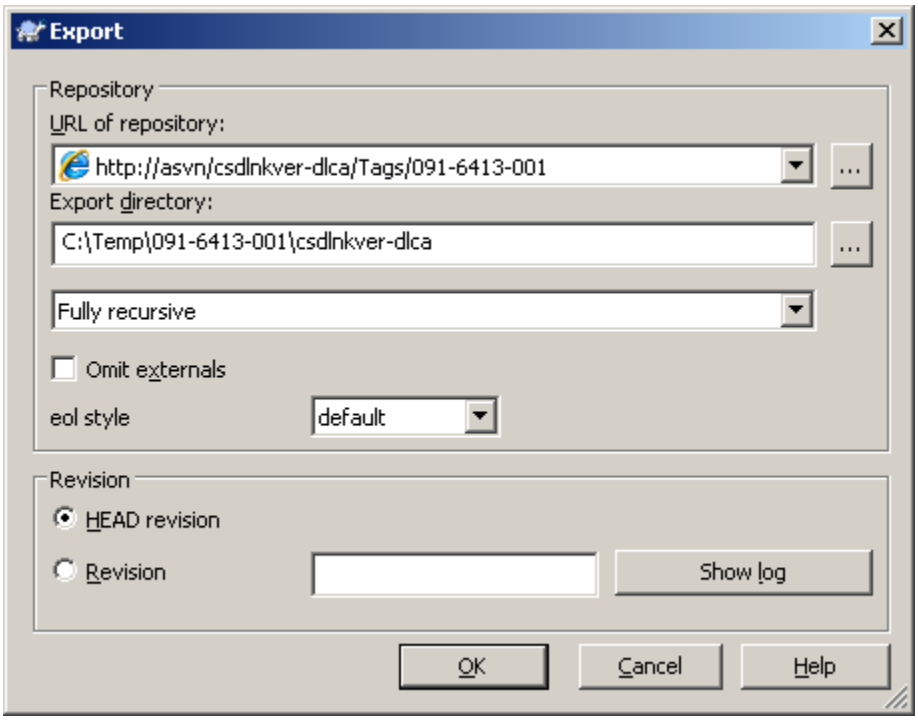


Figure 16: SVN – Archiving the results

Also copy the SCA folder from the code SVN repository at: <Verification-Branch>/sc_results. Then right click on C:\Temp\XXX-XXXX-XXX and Send to → Compressed (zipped) folder. This XXX-XXXX-XXX.zip can now be placed onto the "R:" drive for release into the SCL. Note: R: means \\ccanet\cs\ats\scl.

6 Custom Builds Instructions

6.1 IPS Target Build

1. Check out the latest build(810-0315-XXX) from the PDM tool for the corresponding ***<IPS_Program name>.***
   Note: <IPS_program name> refers to CSeries, GS, KC390_EEJ, M170, MRJ..etc

2. Make sure the latest product line xml files are existed in build at ***<Dev-Branch>*/Configuration\<*IPS_Program name*>**

   DLCA_cfg.xml
   DLCA_iocfg.xml

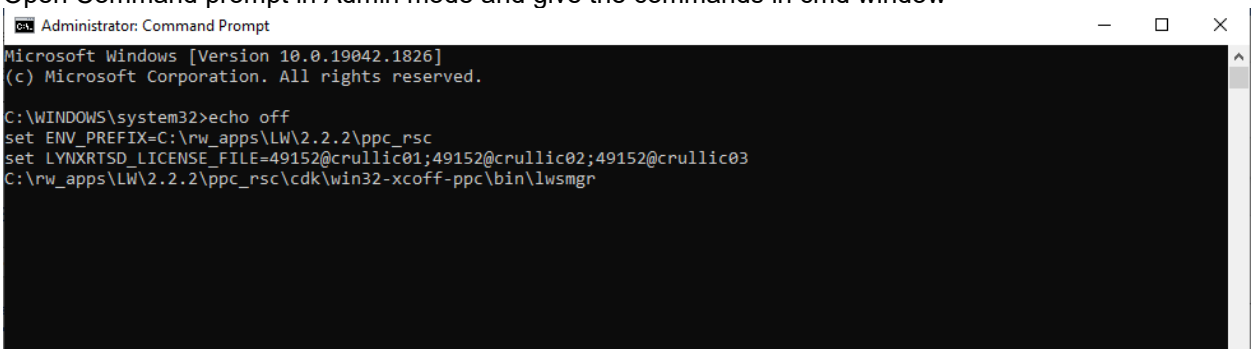
   And DLCA_AtnAppData.xml is exist in build at ***<Dev-Branch>*/Configuration\.**

3. LynxOS-178 v2.2.2 is required to perform a target build using Eclipse. Copy the below commands into a notepad and save it as launch_lwsmgr.cmd.

   echo off
   set ENV_PREFIX=C:\rw_apps\LW\2.2.2\ppc_rsc
   set LYNXRTSD_LICENSE_FILE=49152@crullic01;49152@crullic02;49152@crullic03
   C:\rw_apps\LW\2.2.2\ppc_rsc\cdk\win32-xcoff-ppc\bin\lwsmgr

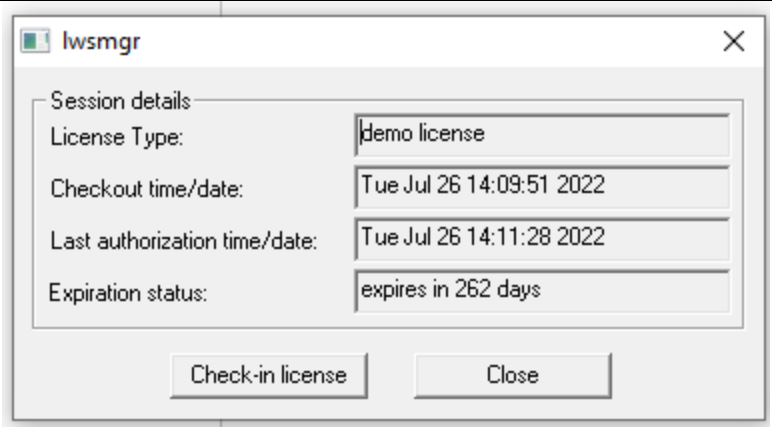
   (or)

   Open Command prompt in Admin mode and give the commands in cmd window

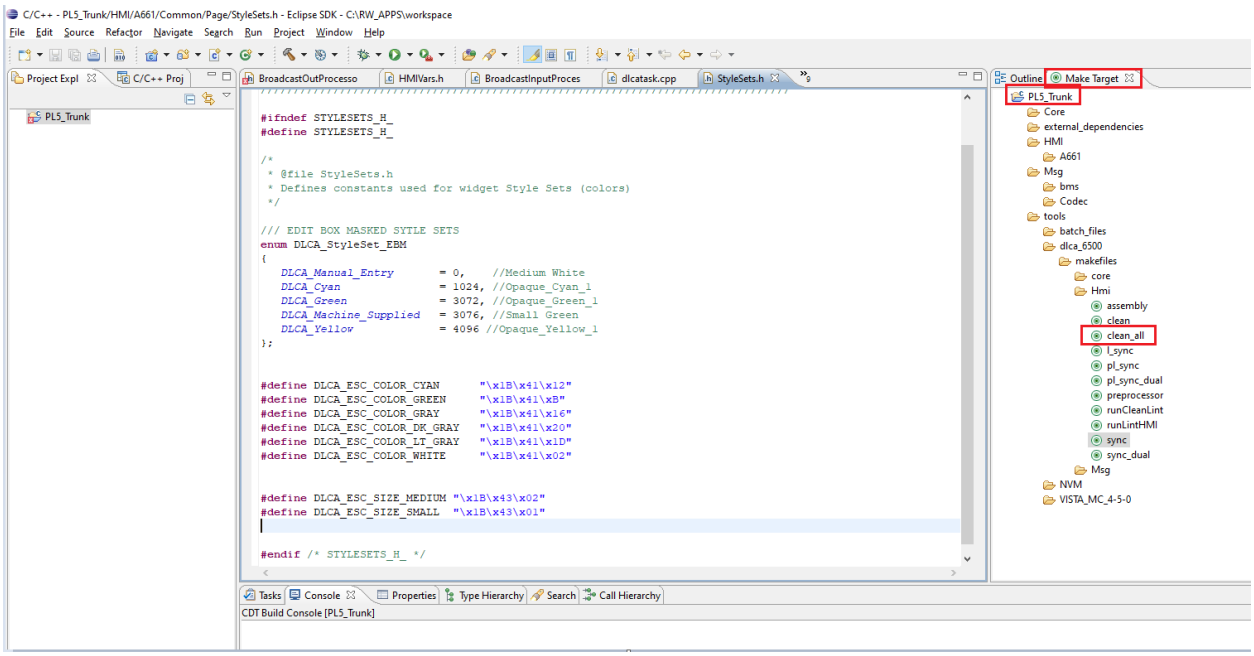


   This cmd file need to be run and keep it open all of the times to perform a target [Power PC (ppc)] build.

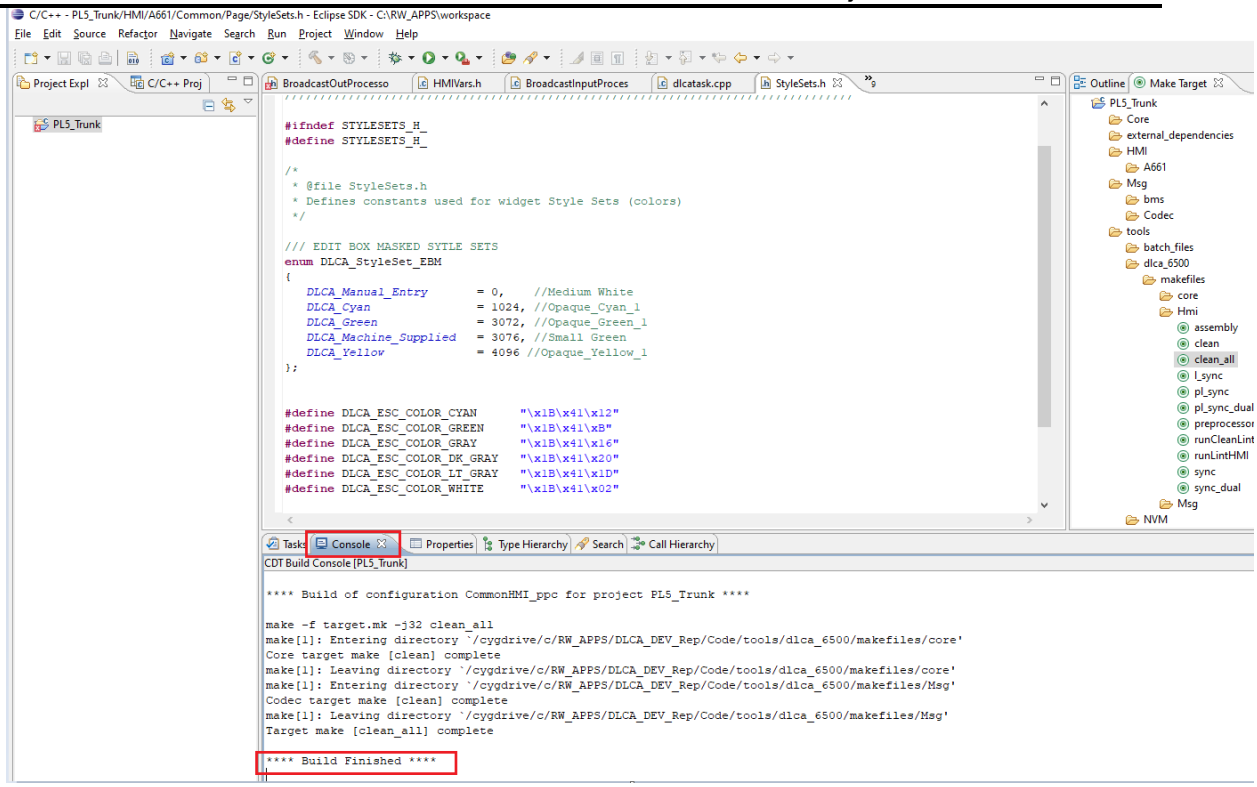
4. Upon launching the batch file the following console will be displayed



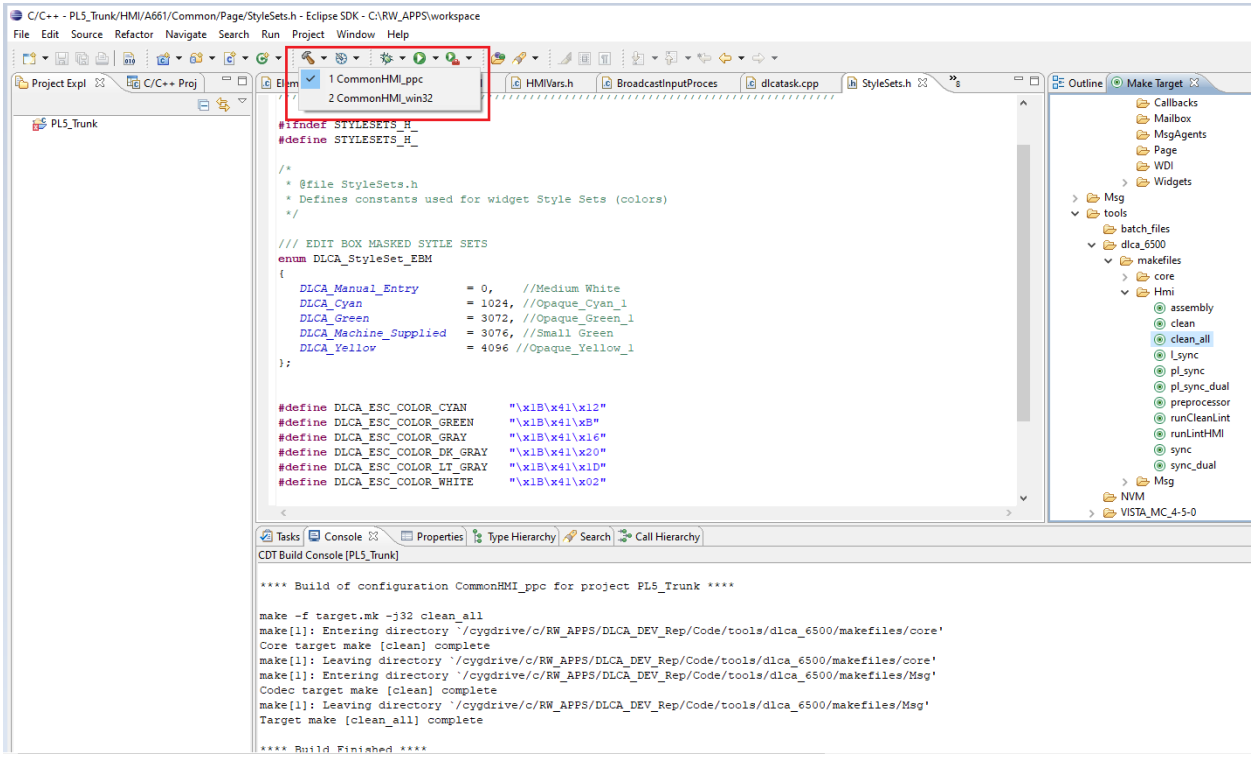
5. Go to Eclipse project, Clean the project (remove all previously created object files), if necessary, by double clicking the clean_all icon for the applicable Makefiles project folder shown in the Make Targets window as shown in below figure



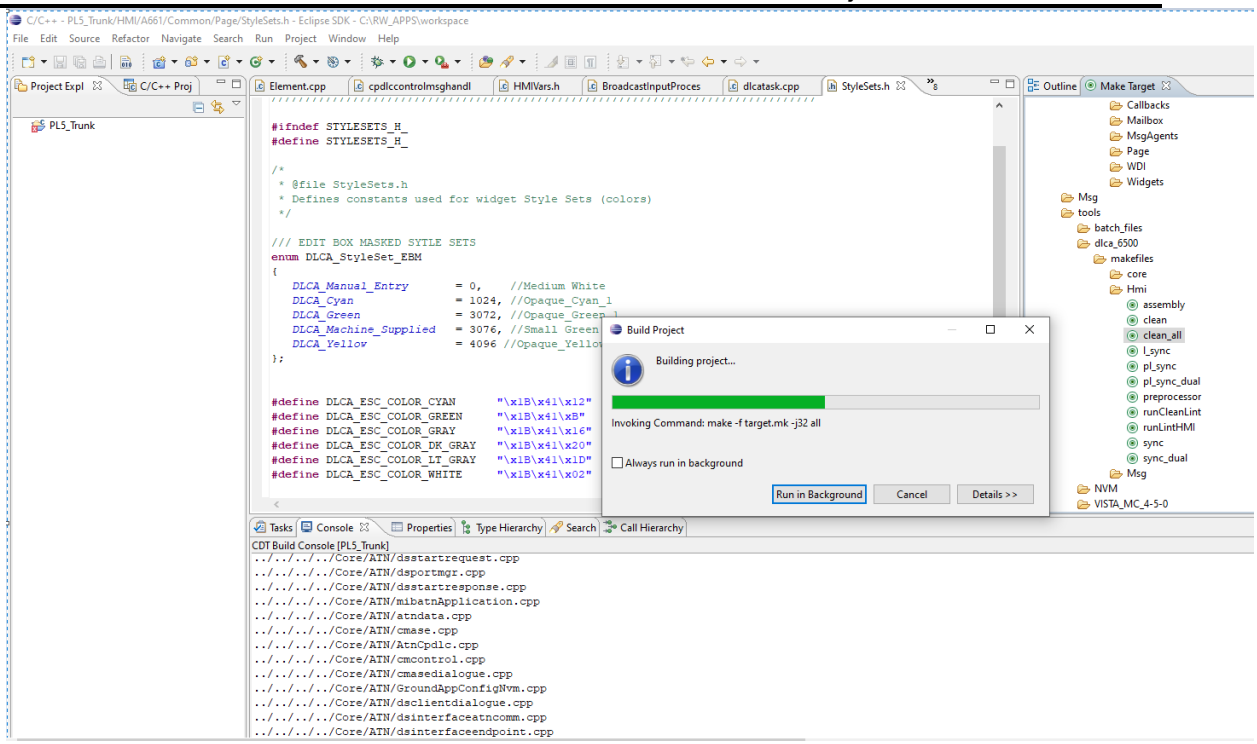
6. Check for the message (**** Build Finished ****) in Console, once the clean all is done.



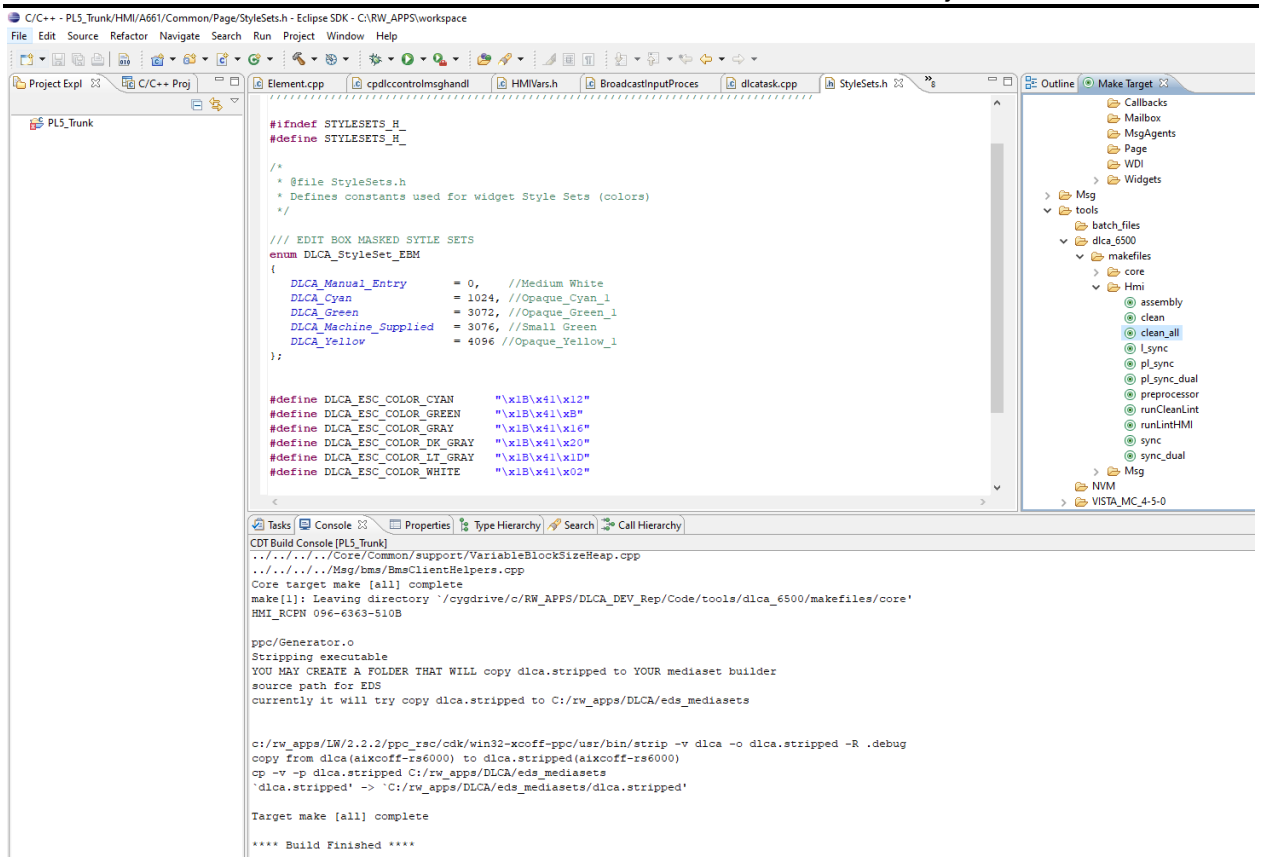
7. Click the 'hammer' icon on the Eclipse menu bar to build dlca application. Select CommonHMI_ppc option to build dlca application.



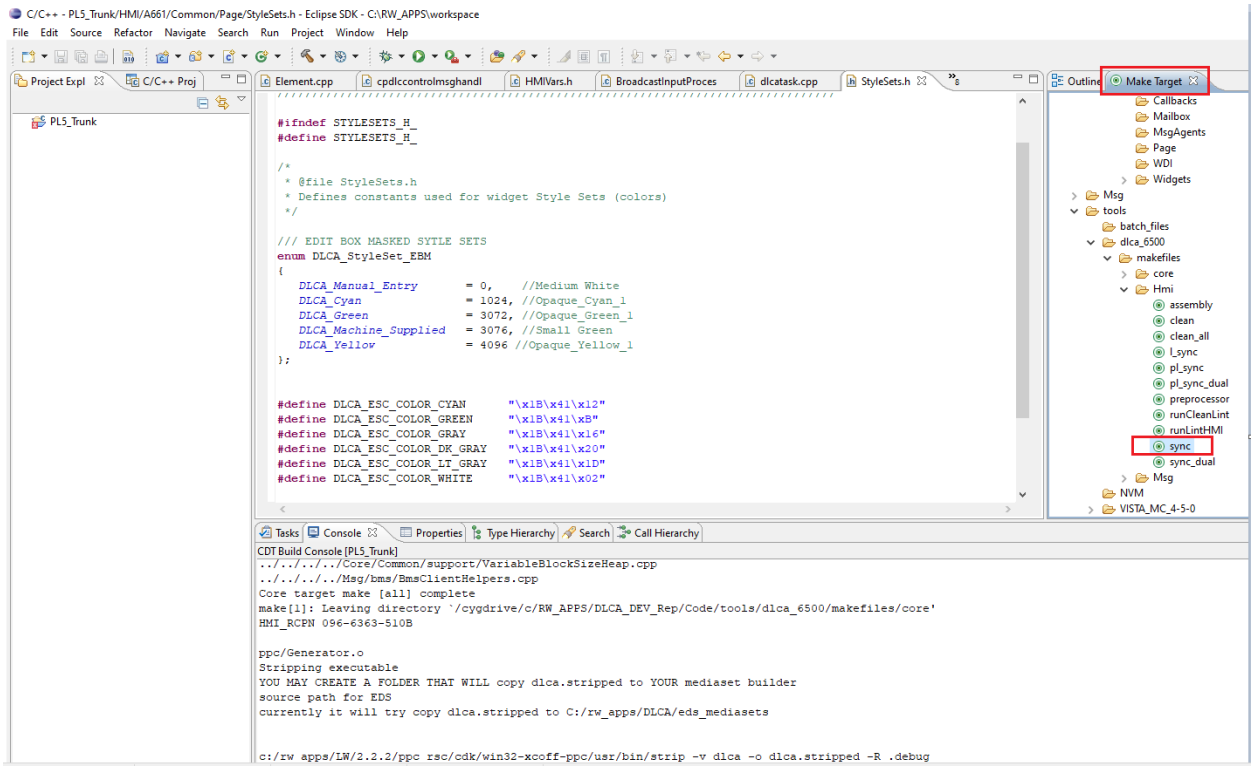
8. Once the build is started, 'Building project' console apperas as shown in below figure.



9. Check for the message (**** Build Finished ****) in Console, once the build is done.



10. Check that dlca and dlca.stripped files are created at ..\Code\tools\dlca_6500\makefiles\Hmi
11. In Eclipse, at 'Make Target' tab , double click on 'sync' option for single DLCA localfs file and sync_dual option for dual DLCA localfs file.



12. Go to ..\Code\tools\dlca_6500\makefiles\Hmi, check that localfs_<IPS_program name> file(s) is created for single DLCA and localfs_L_6500_M170, localfs_R_6500_M170 are created for dual DLCA.

6.2 IPS Custom Builds

6.2.1 IPS Target Build with Invalid XMLs and multiple resets

1. Check out the latest build(810-0315-XXX) from the PDM tool for the corresponding ***<IPS_Program name>.***
   Note: <IPS_program name> refers to CSeries,GS, KC390_EEJ, M170, MRJ..etc

2. Make sure the latest product line xml files are existed in build at ***..\Configuration\<IPS_Program name>***

   DLCA_cfg.xml
   DLCA_iocfg.xml

   And DLCA_AtnAppData.xml is exist in build at ***..\Configuration\.***

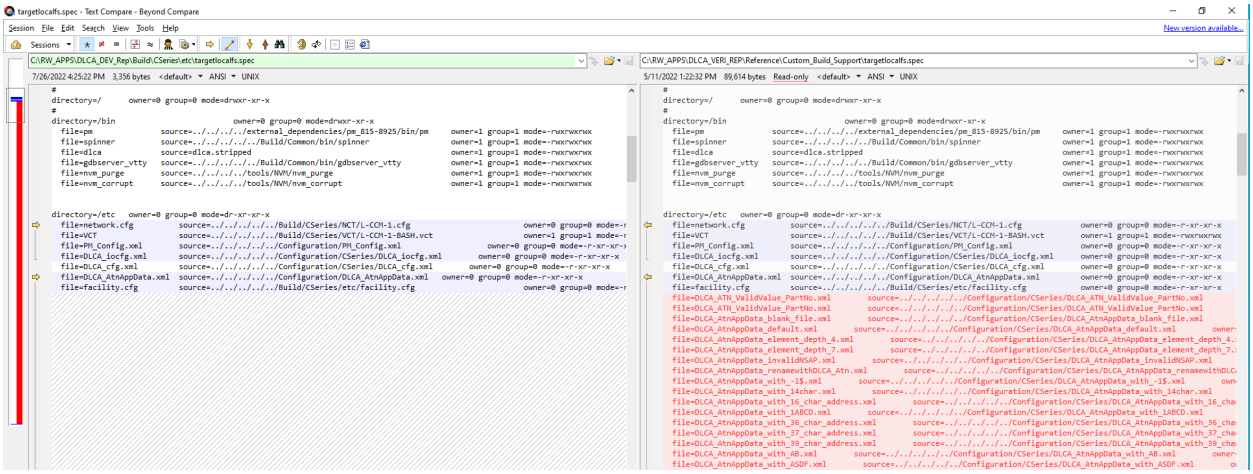
3. Copy all new invalid XML config files from

<Verification-Branch>/tools/Custom_DLCA_Builds/With_Invalid_XML_files/InvalidXML_files

to respective <IPS_Program name> Configuration Directories.

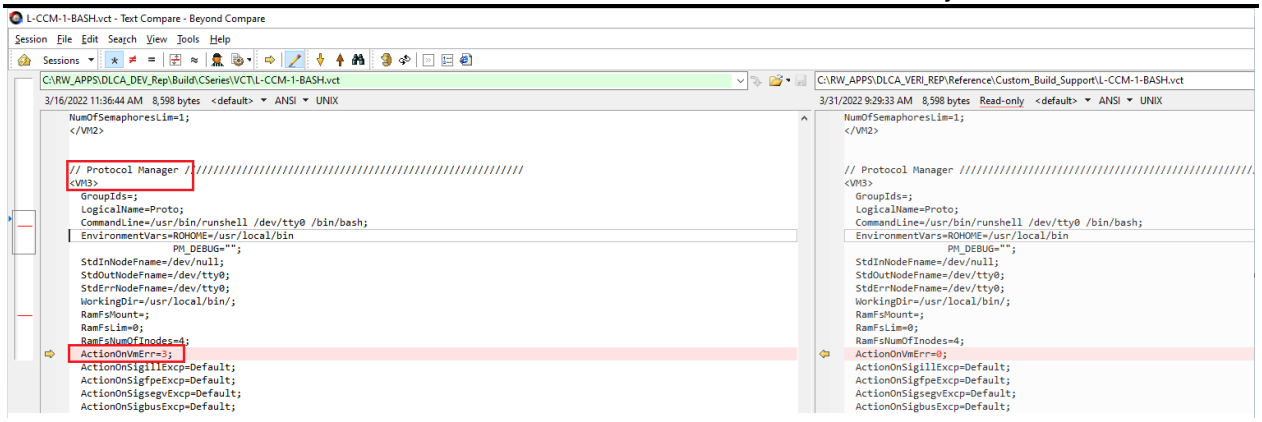
..\Configuration\<IPS_Program name>

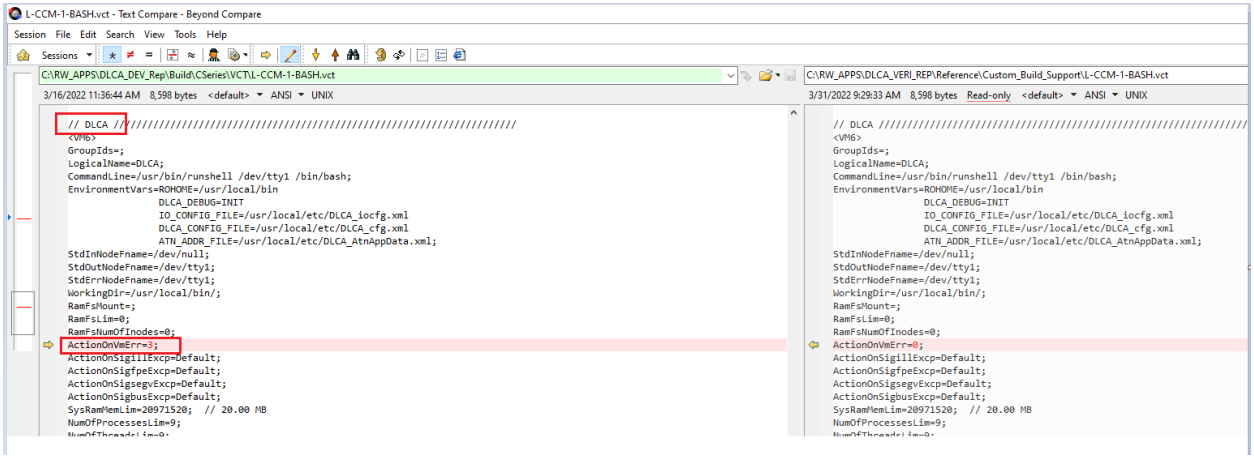
4. Update the respective spec file - targetlocalfs.spec to include all the invalid xml config files at ..\Build\<IPS_Program name>\etc in the latest build



5. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as given below.

   Update 'ActionOnVmErr' to 'Zero' at // Protocol Manager and // DLCA





Follow the steps from 3 to 12 of section – 6.1 Product Line Target Build to generate localfs_<IPS_program name> with invalid XML config files.

Now, localfs_<IPS_program name> contain the custom product line build with invalid XML config files.

6.2.2 IPS Target Build with Invalid Environmental variables

For DLCA_AtnAppData.xml:

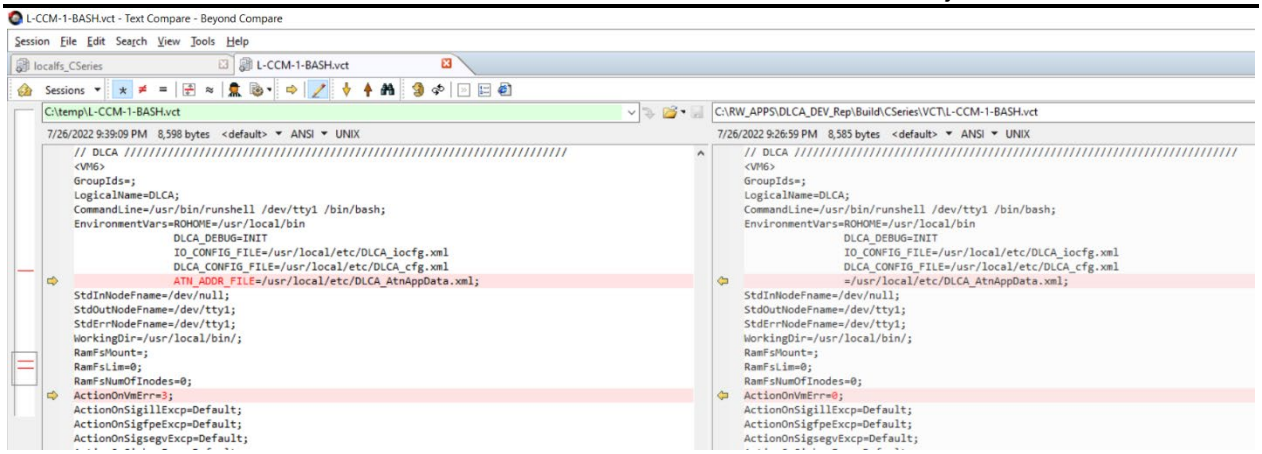
1. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as mentioned below under '// DLCA' at 'EnvironmentVars'.

   From:

   ATN_ADDR_FILE=/usr/local/etc/DLCA_AtnAppData.xml;

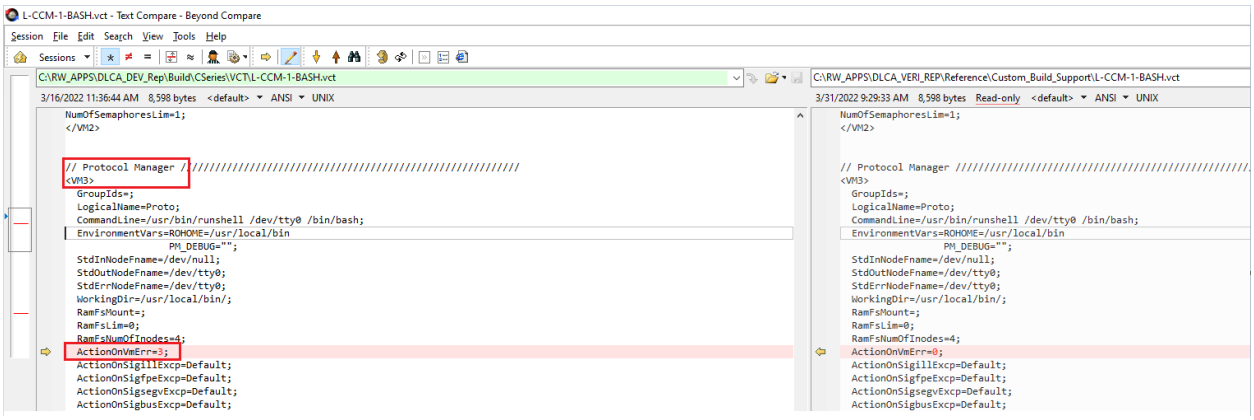
   To:

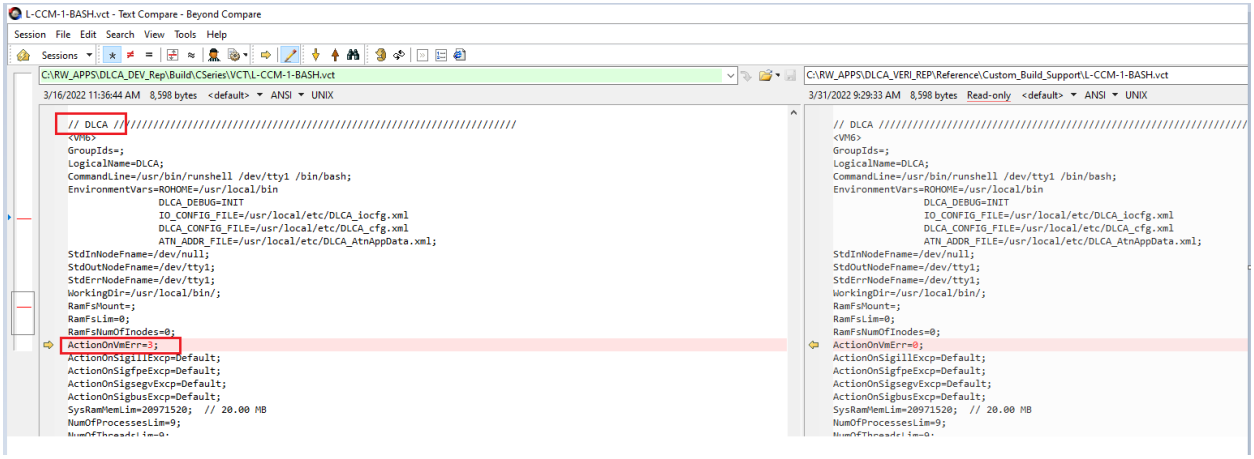
   =/usr/local/etc/DLCA_AtnAppData.xml;



2. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as given below.

Update 'ActionOnVmErr' to 'Zero' under // Protocol Manager and // DLCA





3. Follow the steps mentioned as mentioned in section - 6.1 Product Line Target Build, to generate the localfs_<IPS_program name> for Invalid environmental variables (DLCA_AtnAppData.xml).

For DLCA_cfg.xml:

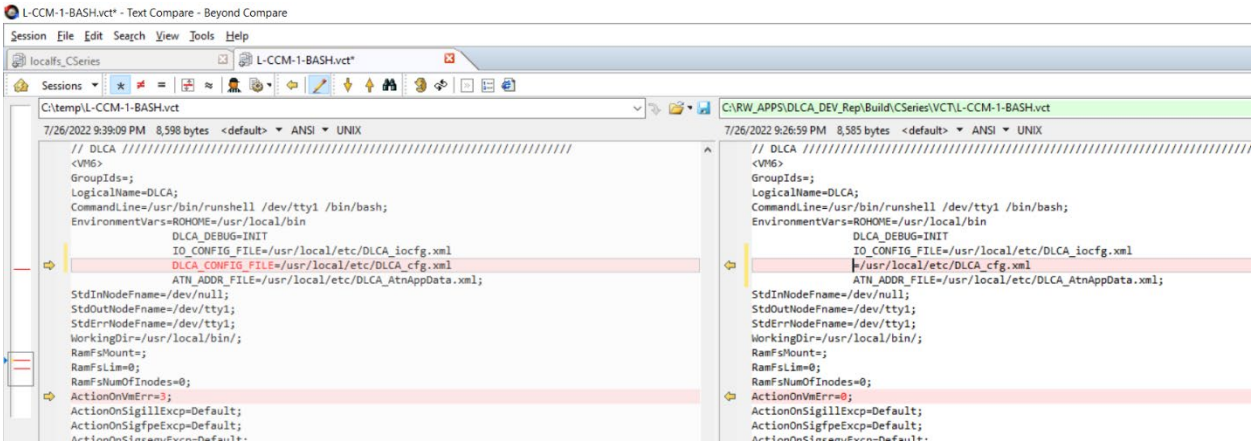
1. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as mentioned below under '// DLCA' at 'EnvironmentVars'.

   From:

   DLCA_CONFIG_FILE=/usr/local/etc/DLCA_cfg.xml

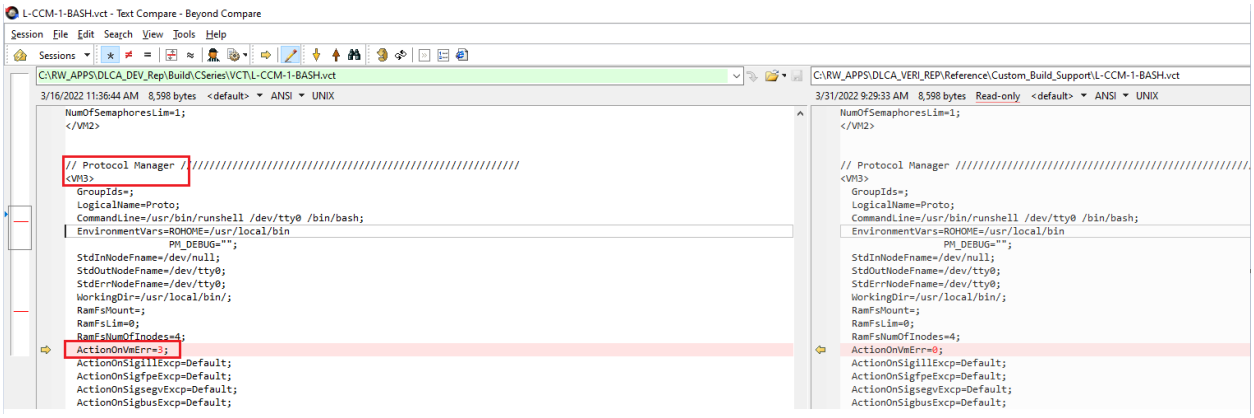
   To:

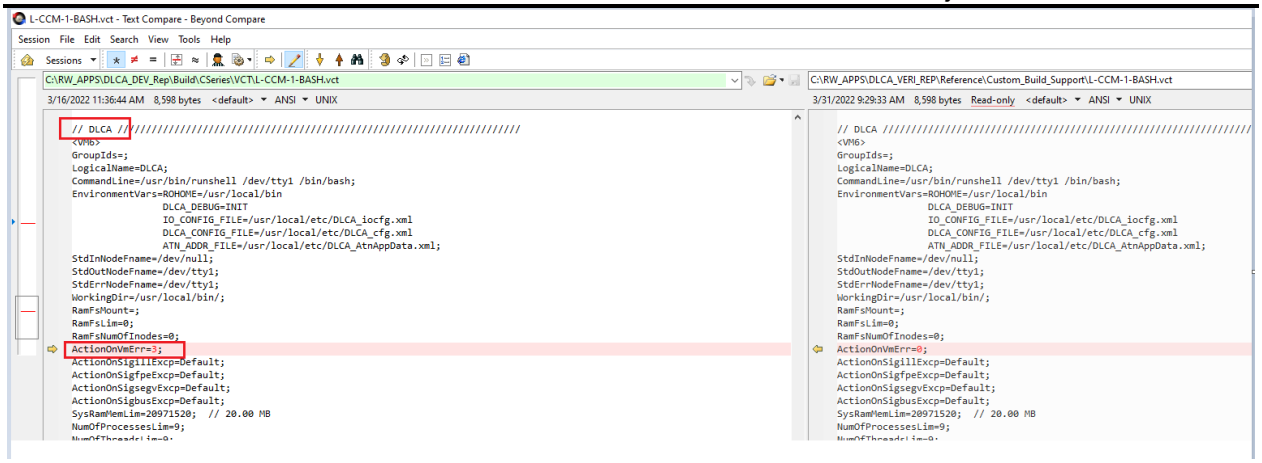
   =/usr/local/etc/DLCA_cfg.xml



2. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as given below.

   Update 'ActionOnVmErr' to 'Zero' under // Protocol Manager and // DLCA





3. Follow the steps mentioned as mentioned in section - 6.1 Product Line Target Build, to generate the localfs_<IPS_program name> for Invalid environmental variables (DLCA_cfg.xml).

For DLCA_iocfg.xml:

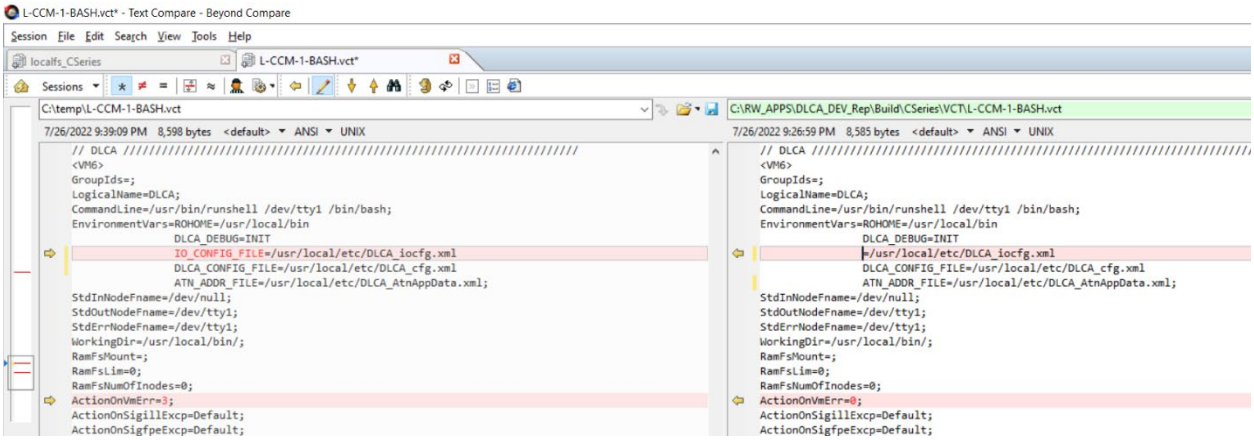
1. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as mentioned below under '// DLCA' at 'EnvironmentVars'.

   From:

   IO_CONFIG_FILE=/usr/local/etc/DLCA_iocfg.xml

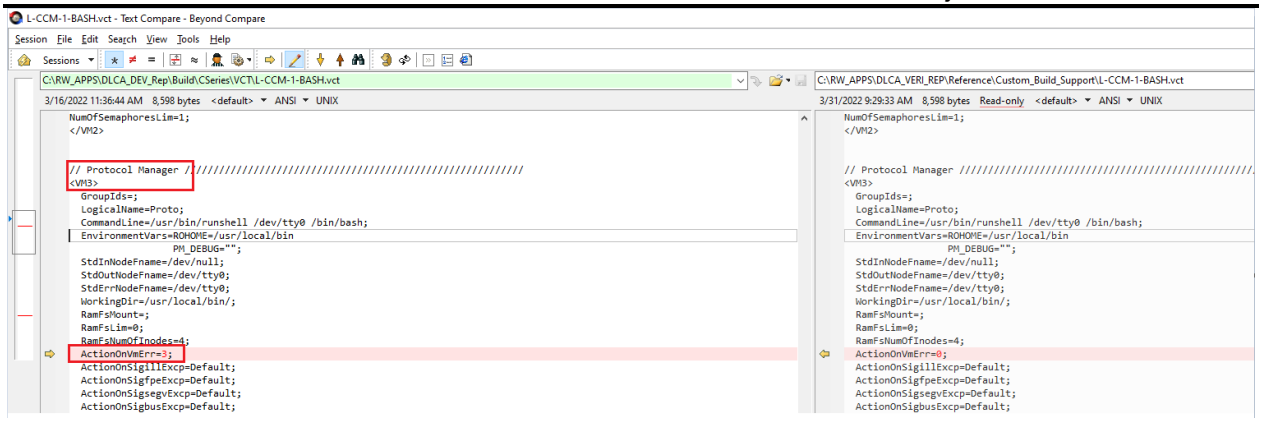
   To:

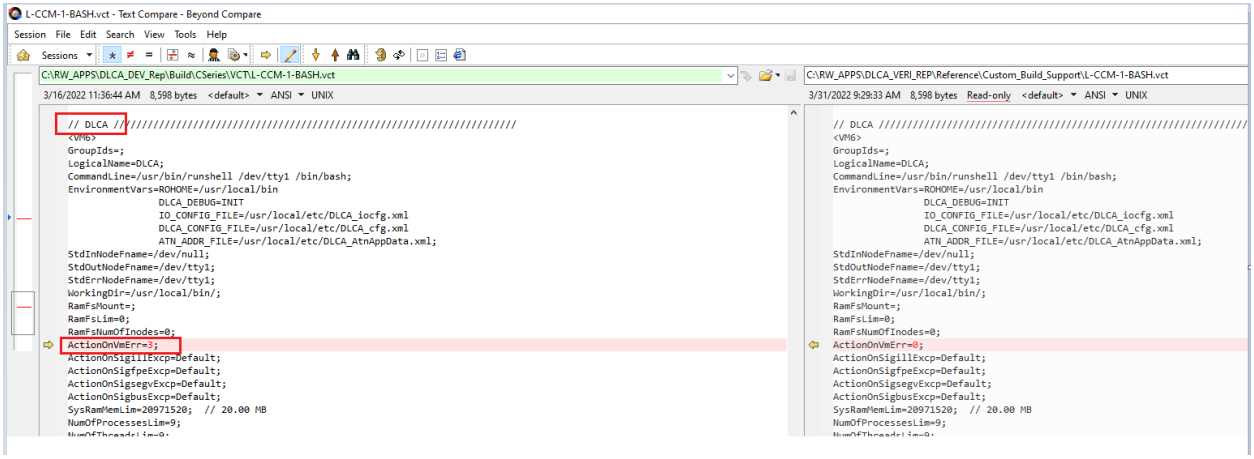
   =/usr/local/etc/DLCA_iocfg.xml



2. Update the L-CCM-1-BASH.vct file at ..\Build\<IPS_Program name>\VCT in the latest build as given below.

   Update 'ActionOnVmErr' to 'Zero' under // Protocol Manager and // DLCA





3. Follow the steps mentioned as mentioned in section - 6.1 Product Line Target Build, to generate the localfs_<IPS_program name> for Invalid environmental variables (DLCA_iocfg.xml).

6.3 Timing Build

The following scripts needs to be executed on special timing build.

- TP_TMNG_001.py
- TP_ACTIVE_STANDBY_005.py
- TP_DMGR_023.py
- TP_CSS_072.py

Follow the below instructions to create a Timing build for IPS

Copy the below files https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/test_procedures/TIMING and replace in the respective folder structure in latest build from PDM (Merge the files properly to as per folder structure).

- dlcatask.cpp
- dlcatask.h
- ErrorManager.cpp
- ErrorManager.h
- pmBopInterface.cpp
- Dmc.cpp
- debug.cpp
- debug.h
- adsreport.cpp
- cdsConnectionManager.cpp

- MessageArea.cpp
- Include_target.mk

After merging files, Follow the steps mentioned in section – 6.1 Product Line Target Build to generate localfs_<IPS_program name> for timing build.

NOTE: Test procedures that require such custom build(s) will be marked in the TCS file via the 'Remarks' column.

Follow the below instructions to create a Timing build for EDS:

1. Copy the below files https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/test_procedures/TIMING and replace in the respective folder structure in latest build from PDM (Merge the files properly to as per folder structure).
   - dlcatask.cpp
   - dlcatask.h
   - ErrorManager.cpp
   - ErrorManager.h
   - pmBopInterface.cpp
   - Dmc.cpp
   - debug.cpp
   - debug.h
   - adsreport.cpp
   - cdsConnectionManager.cpp
   - MessageArea.cpp
   - Include_target.mk

2. After merging files, Generate the dlca.stripped file as per steps mentioned in section – 6.1 Product Line Target Build (until Step 10)

3. Get the dlca.stripped file to create Mediaset for timing build follow the steps mentioned at section - 6.4 EDS Target Build.

6.4 EDS Target Build

1. Check out the latest build(810-0315-XXX) from the PDM tool for the corresponding ***<EDS_Program name>.***
   Note: <EDS_program name> refers to EDS_C295, EDS_CL604, EDS_M204..etc

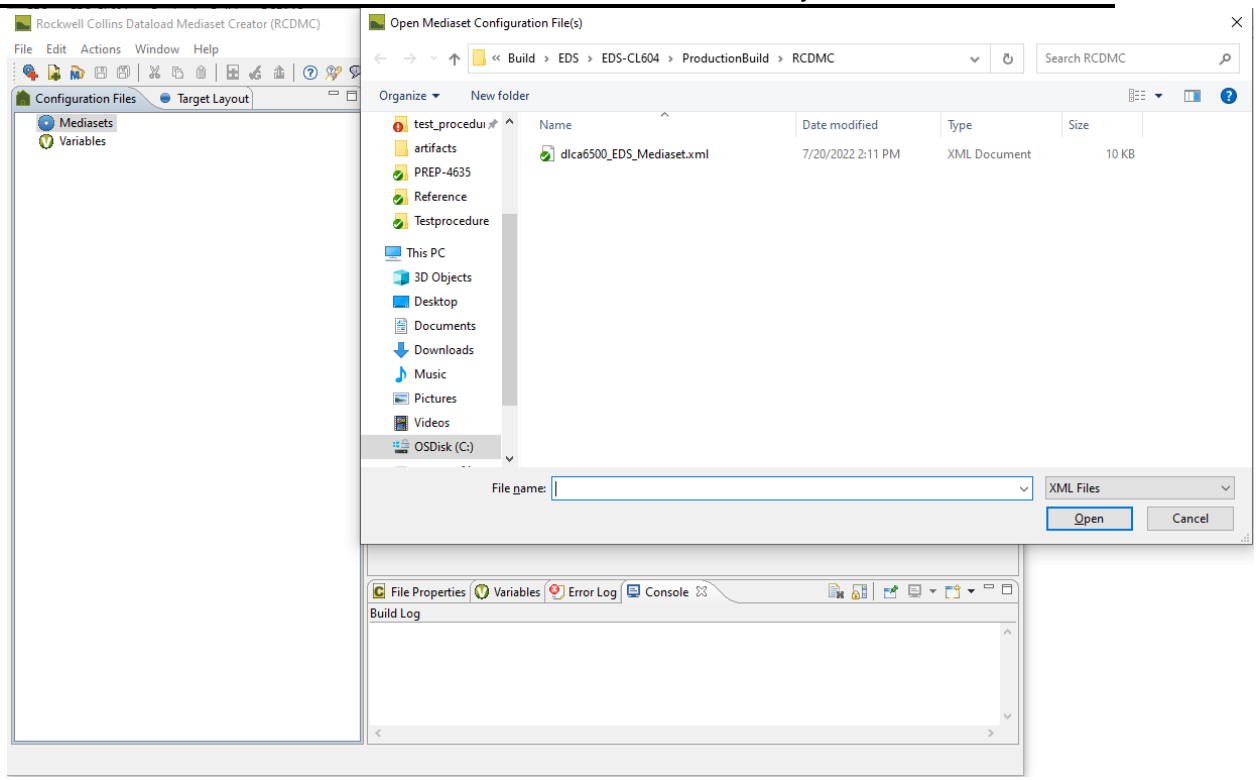
2. Get the dlca.stripped file from PDM tool (as per step 1) and place at below location ***..\Build\EDS\dlcaExec***

3. To create Mediaset, Go to the path ***..\Tools\RCDMC***

   a. Open the Rockwell Collins Dataload Mediaset Creator, i.e, Double click on 'RCDMC.exe' tool

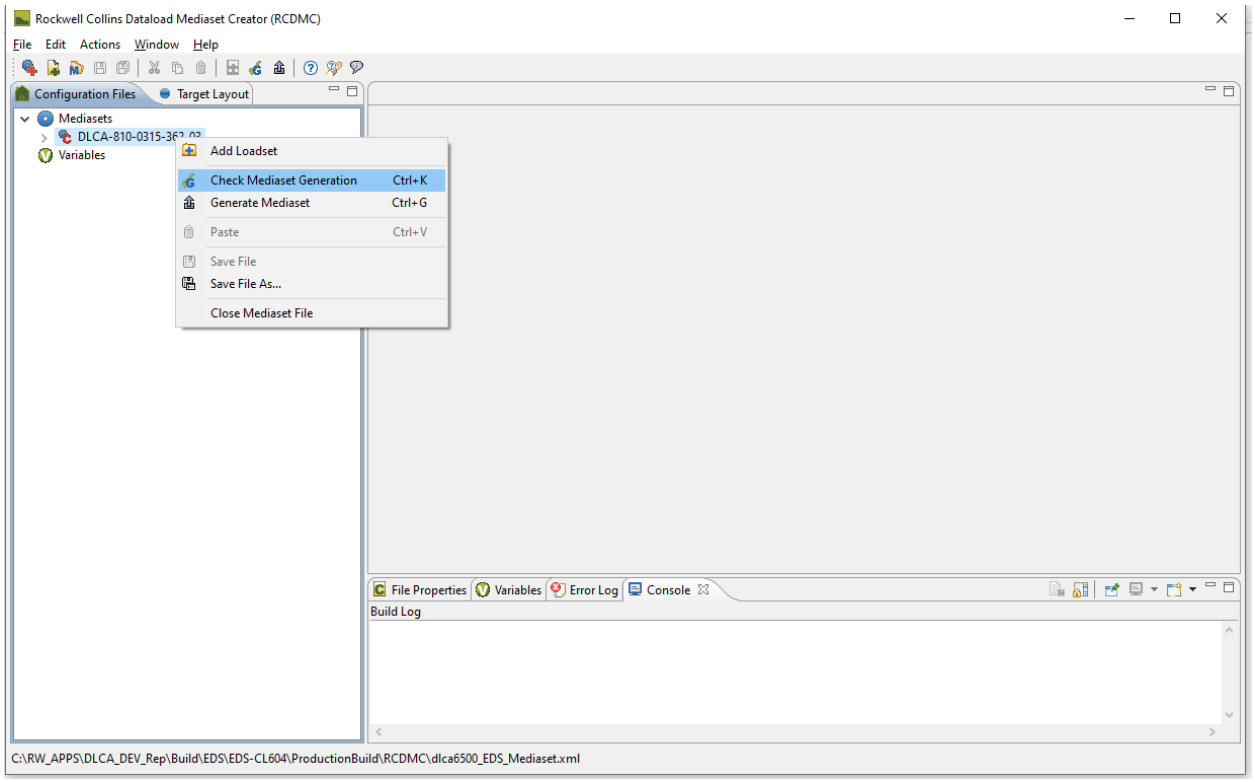
   b. Once the RCDMC tool is opened, File -> Open -> Open Mediaset Config File

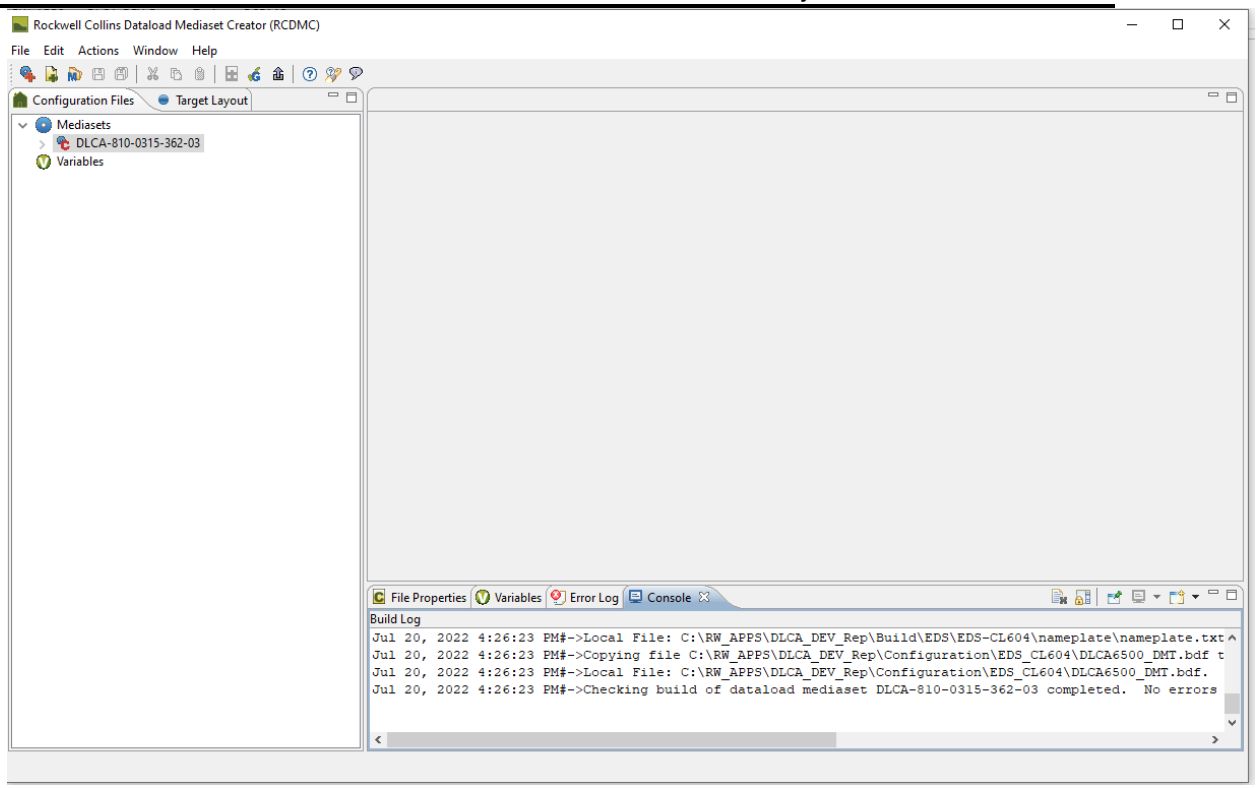
   c. Browse the file 'dlca6500_EDS_Mediaset.xml' from **..\Build\EDS\<EDS_Program name>\ProductionBuild\RCDMC** as shown in the below figure(Example).



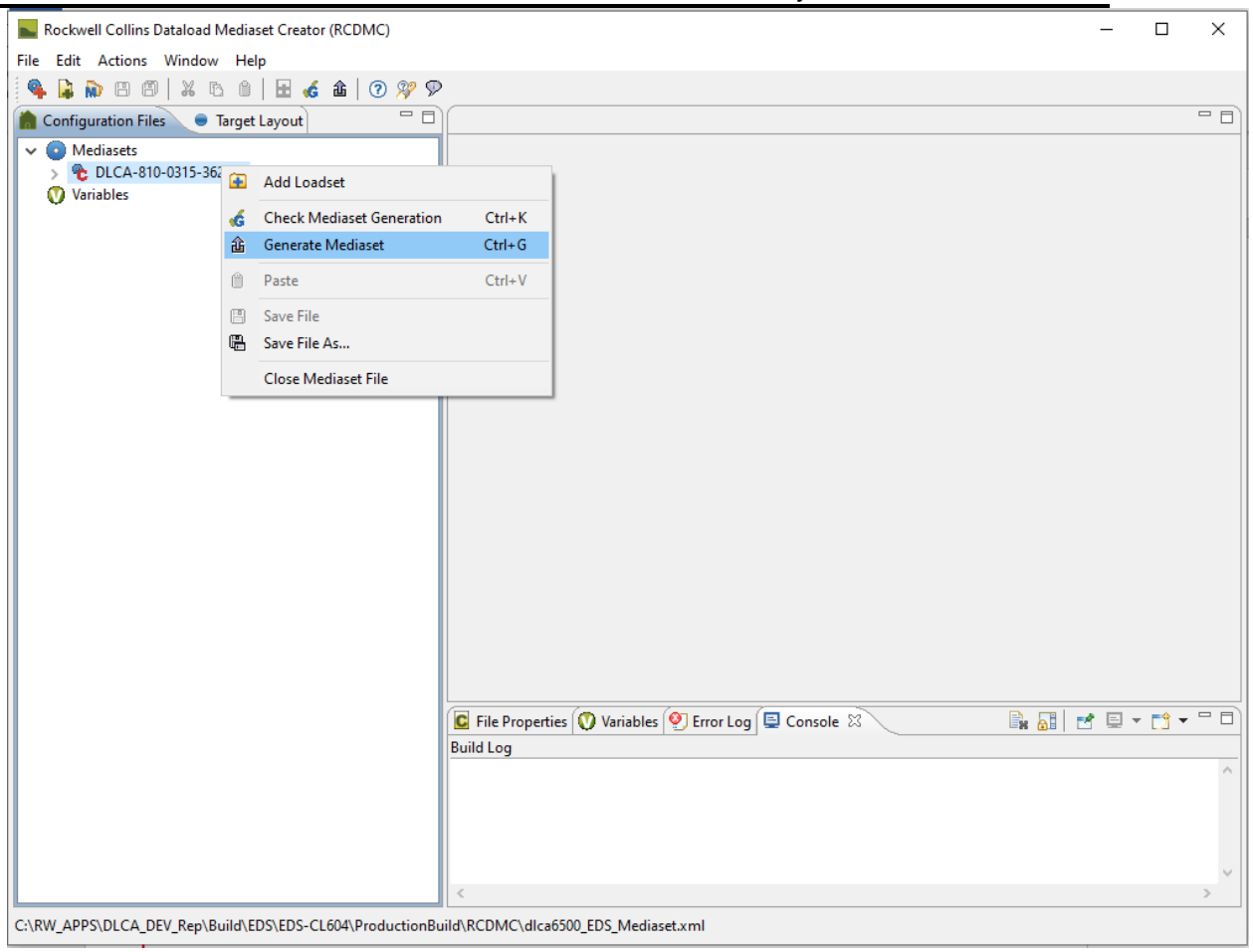
d. Click on Open

e. Go to Mediasets option and then right click on Mediasets after that select Check Mediaset Generation option and address any issues shown in the Console or Errors tabs. If no errors are generated, create the mediaset

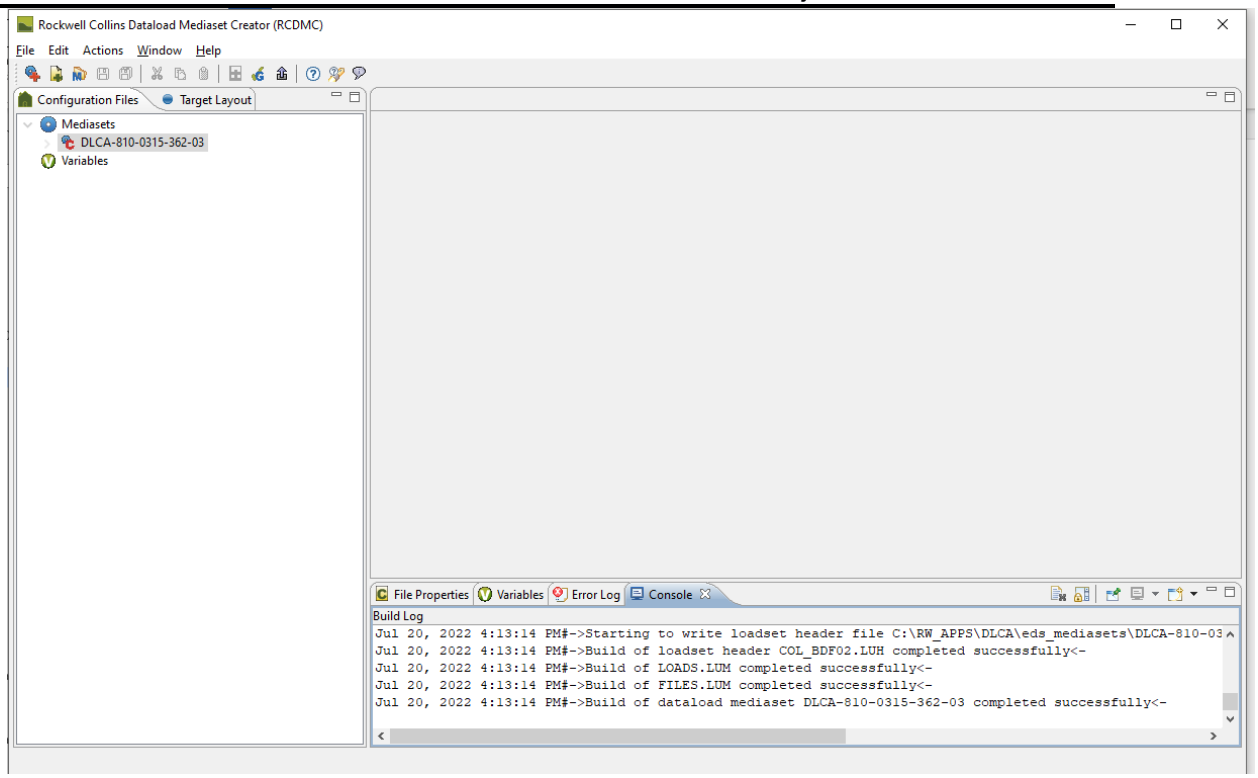




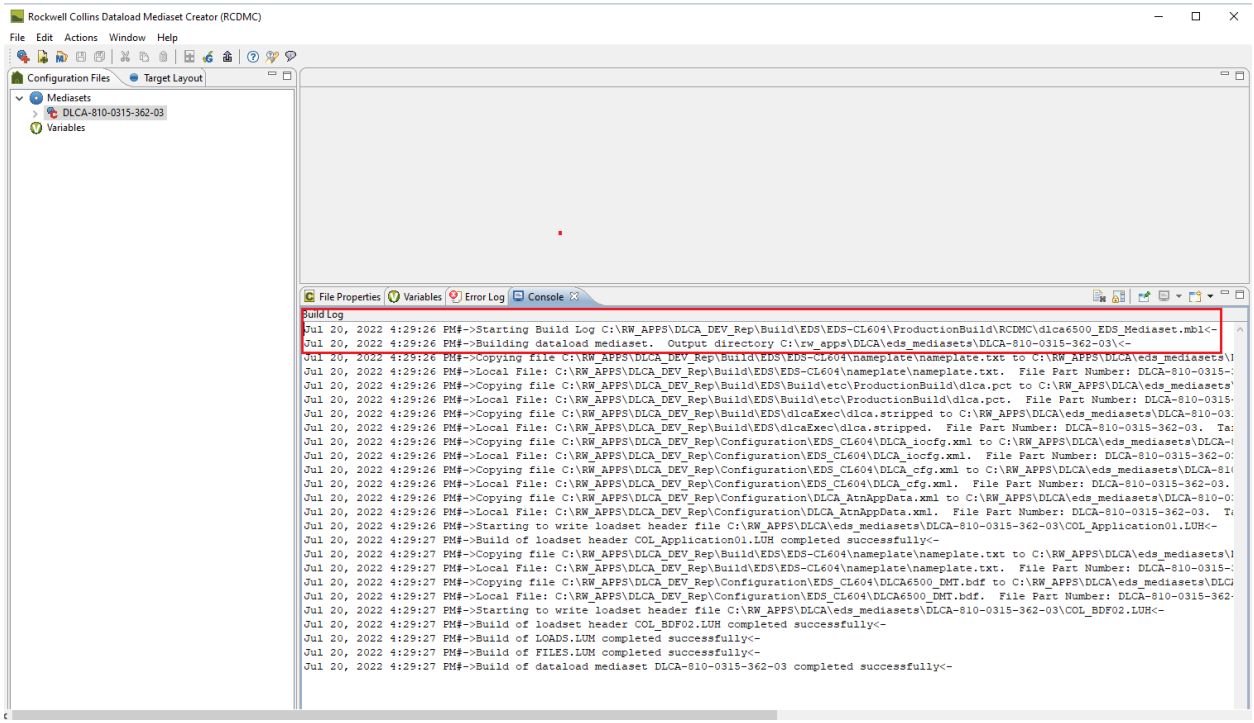
f. If no errors, Go to Mediasets option and then right click on Mediasets after that select Generate Mediaset option otherwise fix the errors and generate the mediaset.



g. Check in the Console window Mediaset is generated successfully or not.



h. Go to the Build log path and get the Mediaset to load on Target



6.5 EDS Target Build with Invalid XML files

1. Check out the latest build(810-0315-XXX) from the PDM tool for the corresponding ***<EDS_Program name>.***
   Note: <EDS_Program name> refers to EDS_C295, EDS_CL604, EDS_M204..etc

2. Get the dlca.stripped file from PDM tool (as per step 1) and place at below location ***..\Build\EDS\dlcaExec***
3. Make sure below xml files are exist in build folder at ..\Configuration\<EDS_Program name>

   DLCA_cfg.xml

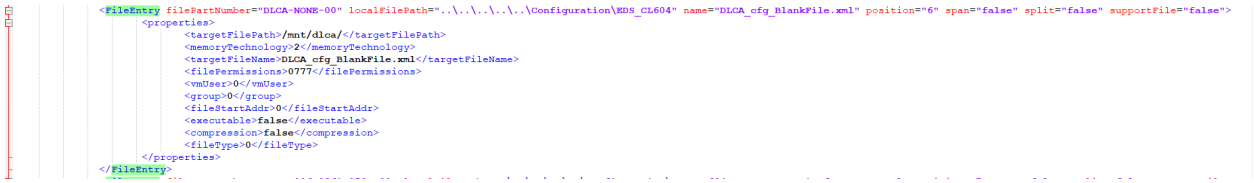
   DLCA_iocfg.xml

   And DLCA_AtnAppData.xml exist in build folder at ..\Configuration\
4. Copy all the invalid xml files from below path

   https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/tools/Custom_DLCA_Builds/With_Invalid_XML_files/InvalidXML_files

   to

   ..\Configuration\<EDS_Program name>
5. Collect the Part Numbers for all the xml files and update the XML file - 'dlca6500_EDS_Mediaset.xml' at **..\Build\EDS\<EDS_Program name>\ProductionBuild\RCDMC** of the following structure for each invalid XML file.



   Update the below FileEntry tag fileds as per program accordingly.

   filePartNumber -> Part number of the xml file

   localFilePath -> File path of the xml file

   name -> name of the xml file

   position -> number of xml file(s) in incremental order

   targetFileName -> name of the xml file

   Consider other tag fields as is mentioned in above figure.

   Note: Append all the invalid xml files data after DLCA_AtnAppData.xml FileEntry tag.
6. Follow the steps mentioned at Step 3 of section - 6.4 EDS Target Build to create mediaset for custom build

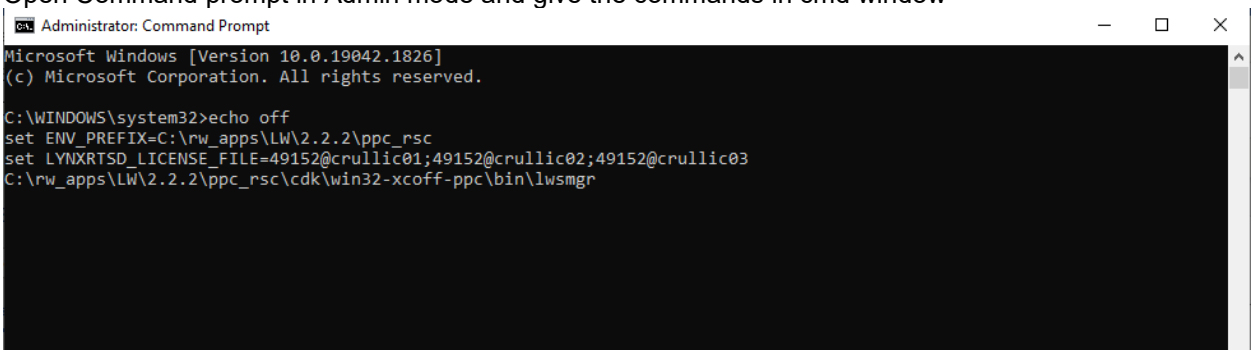
7 Codec Build Instructions

1. 1. LynxOS-178 v2.2.2 is required to perform a target build using Eclipse. Copy the below commands into a notepad and save it as launch_lwsmgr.cmd.

   echo off
   set ENV_PREFIX=C:\rw_apps\LW\2.2.2\ppc_rsc
   set LYNXRTSD_LICENSE_FILE=49152@crullic01;49152@crullic02;49152@crullic03
   C:\rw_apps\LW\2.2.2\ppc_rsc\cdk\win32-xcoff-ppc\bin\lwsmgr

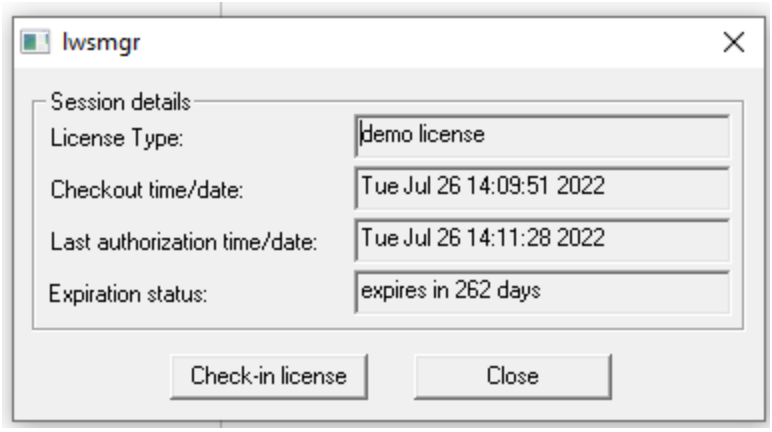
   (or)

   Open Command prompt in Admin mode and give the commands in cmd window

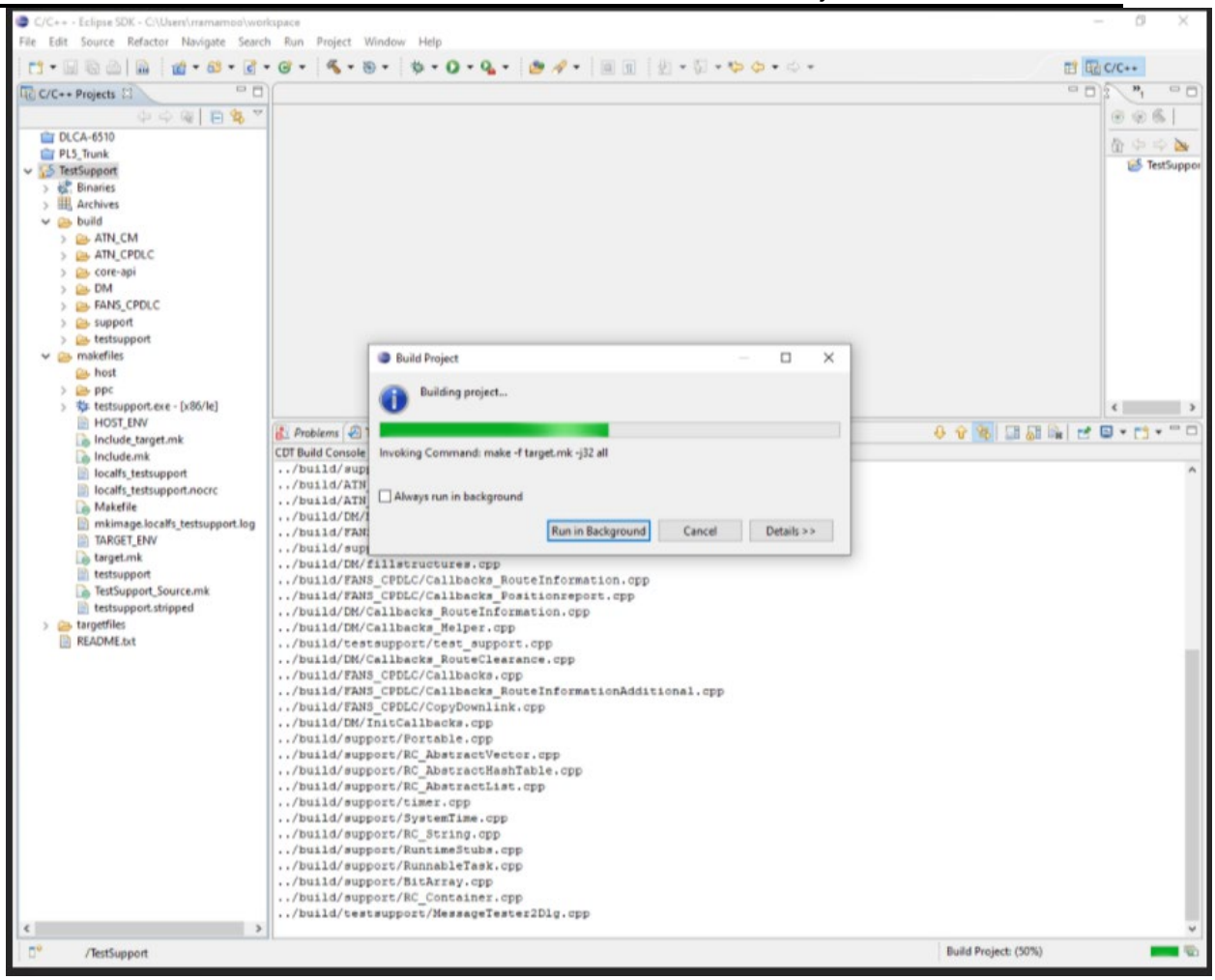


   This cmd file need to be run and keep it open all of the times to perform a target [Power PC (ppc)] build.

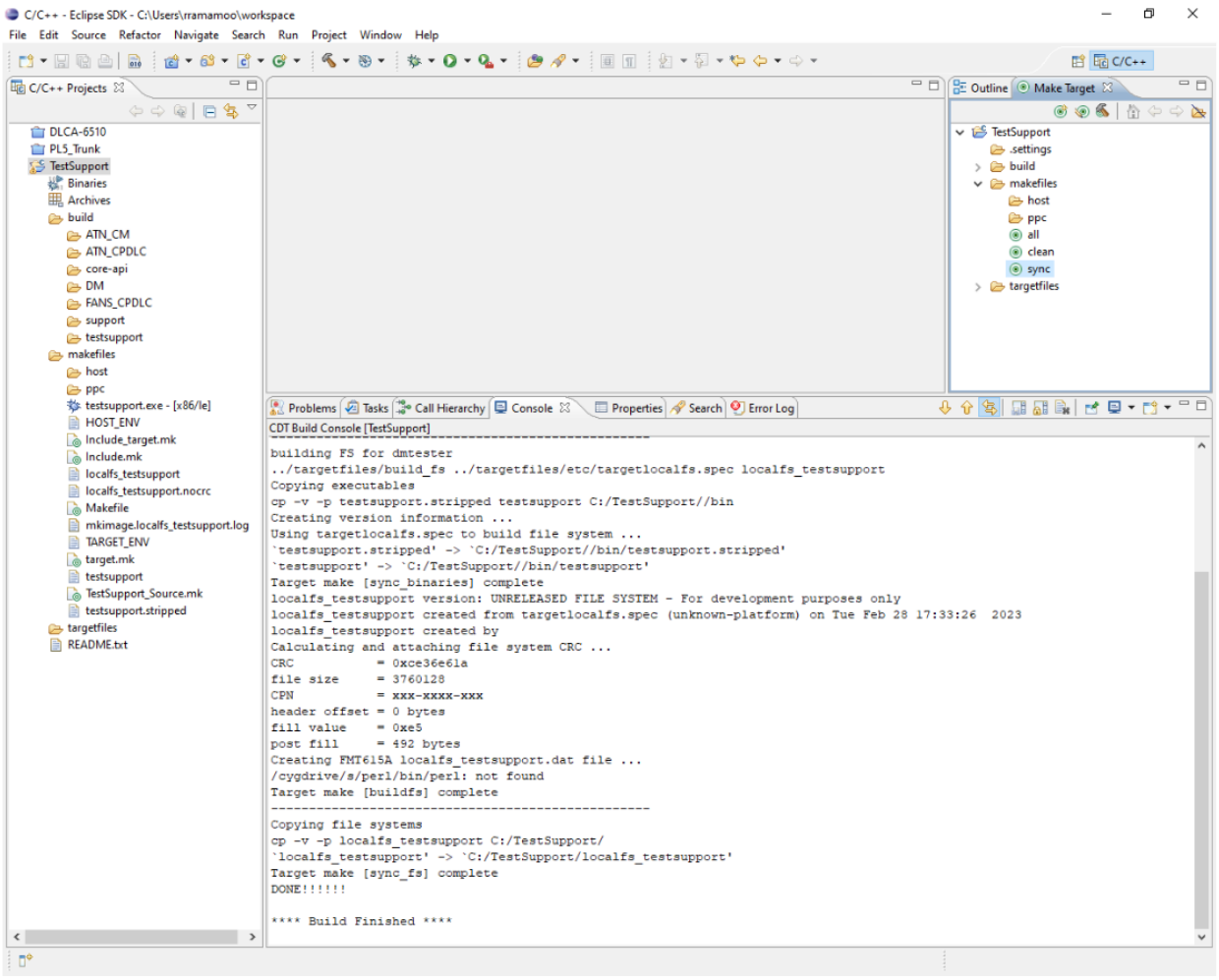
2. Upon launching the batch file the following console will be displayed



3. Import ' <Dev-Branch>/Tools/Harmonized Message Tester' into eclipse.
4. Create the following folder C:\TestSupport, in order to place localfs_testsupport file for codec target build.
5. Go to Eclipse project, Clean the project (remove all previously created object files), if necessary, by double clicking the clean_all icon for the applicable Makefiles project folder shown in the Make Targets window.
6. Click the 'hammer' icon on the Eclipse menu bar to build dlca application. Select CommonHMI_ppc option to build TestSupport application; Once the build is started, 'Building project' console appears as shown in below figure



7. In the Make Targets tab in Eclipse, double click the 'sync'.



8. Check that "localfs_testsupport" file is created at C:\TestSupport\

Note: For dataload steps for codec target, follow steps as in section 8.4.1, and use "localfs_testsupport", while browsing for local fs file to load in step 10 of section 8.4.1.

8 Test Equipment

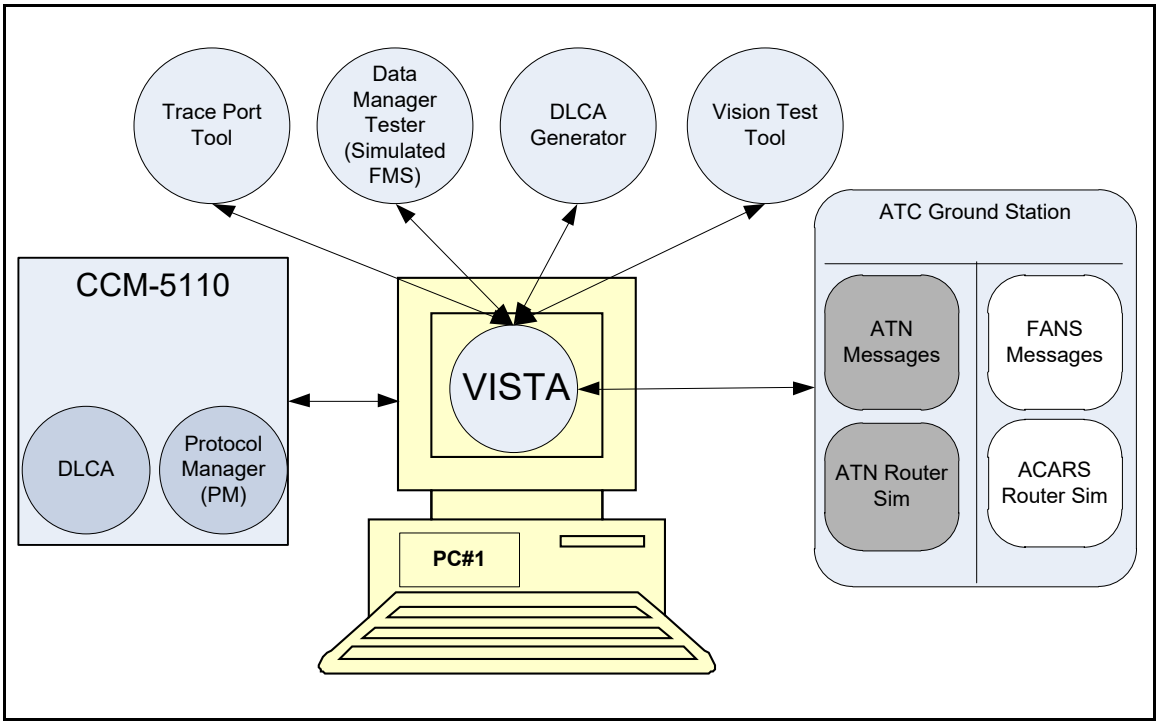


Figure 17: Test Equipment

8.1 VISTA Simulator

8.1.1 Overview of VISTA

VISTA (the Virtual Integrated Software Test bed for Avionics) is a comprehensive, flexible avionics software test environment. It is intended to be used both as an aid to development and as a formal verification tool. At the core of VISTA is a sophisticated simulation of a model aircraft, several cockpit displays and control panels, and the data buses required to connect an array of cockpit devices.

This simulation is controlled by means of Python Scripting. Python commands may be entered at the command line prompt during interactive test sessions, or stored in files to be run in batch mode. Such command files may be combined to form a complete regression test suite.

The Vista environment and its applications are moved into SVN rather than installing. This process minimizes the setup effort.

8.1.2 Features in VISTA

VISTA has been designed to support many aspects of debugging and testing avionics software. The following features demonstrate VISTA's adaptability to the needs of the individual developer or verifier:

- VISTA provides a host–based environment for testing embedded avionics systems by simulating hardware such as serial buses, timers and discrete.
- Simulated buses may be connected to actual hardware buses, with bus data transferred to and from the buses in real time
- Creation of an automated regression test suite is made possible by the Python Scripting and Vision Test Tool, which includes such features for automated testing as simulating pilot actions, simulating failures in avionics, and automatically checking results.

- The tests developed to exercise the avionics application running on a host workstation can subsequently be run against the target hardware with no changes to the test script.
- VISTA consists of a collection of independent programs that run simultaneously under the control of a simulation clock.
- The graphical cockpit displays facilitate debugging of avionics software

There are also other programs available for use with VISTA, like Signal Analyzer, which can be used for monitoring ARINC bus traffic.

8.1.3 Operation of VISTA

VISTA is structured as a collection of programs that are run simultaneously as separate processes under a multi-tasking operating system. These processes are controlled and synchronized by a central process, called V_Control, created when the command is given to initiate the VISTA session. When a command is issued, V_Control routes it to the appropriate process to handle it. V_Control receives error and informational messages from each process and displays them to the user. VISTA processes communicate with each other via simulated buses, discrete, and shared memory areas.

Due to the synchronization performed by VISTA's centralized control process, when execution of an application is stopped, at a debugger breakpoint, for example, the entire simulation is suspended, thus eliminating problems with re-synchronization of the application with the environment once execution resumes.

The open, modular architecture of VISTA allows new programs to be written and integrated into the set of processes governed by the central control process, V_Control, to expand the number of cockpit subsystems exercised using the simulation–based test facility.

How does the modular structure of VISTA affect you as a user of this environment? Since each process is invoked separately by issuing a CREATE command, during your VISTA testing session, you may start up as many or as few of the VISTA processes as you need. For example, if you only want to examine a software function that is observable on the AFD, you simply enter create ('afd_app_left') to present the simulated AFD on your workstation screen. In this case, there is no need for you to issue commands to create the other simulated displays. You may of course give a series of CREATE commands to build a full, dual, simulated cockpit.

8.1.3.1 Initiating the VISTA Session

Vista session can be started by executing the file Lauch_Vista.exe from the below local checkout copy:

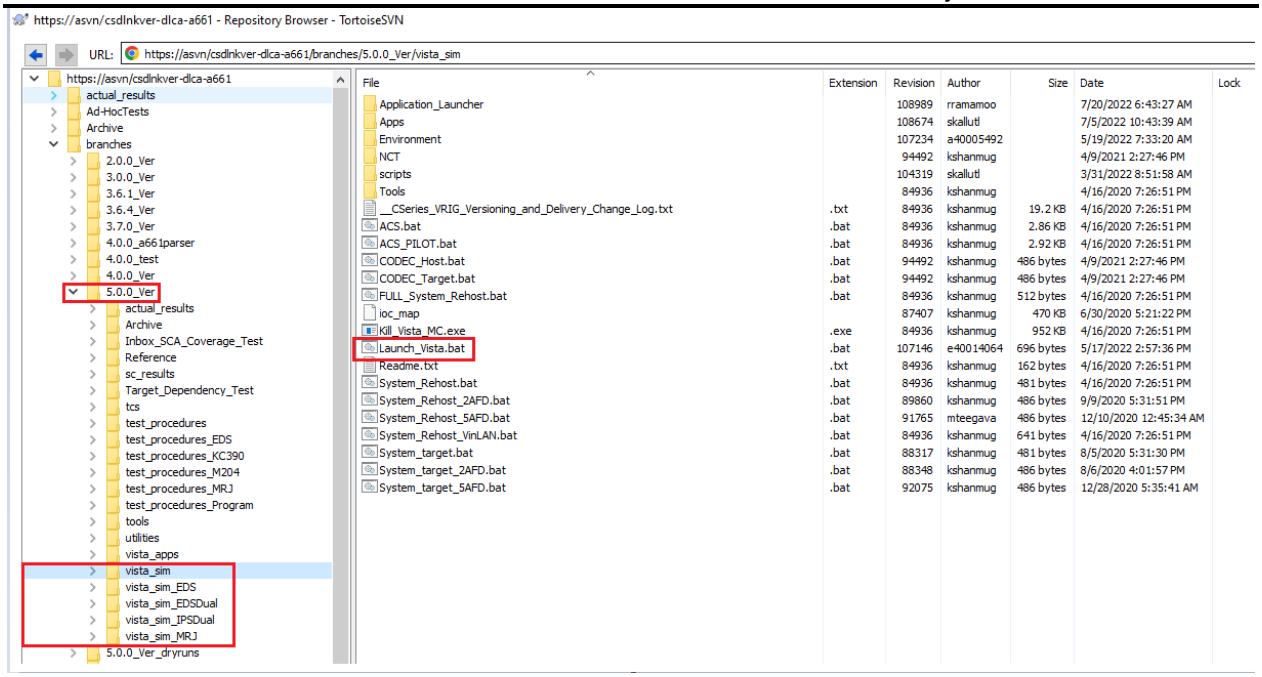
https://asvn/csdlnkver-dlca-a661/branches/<X.X.X_Ver>/<Process_Specific>

Note:

a. Here, <X.X.X_Ver> refers to latest version of verification folder.

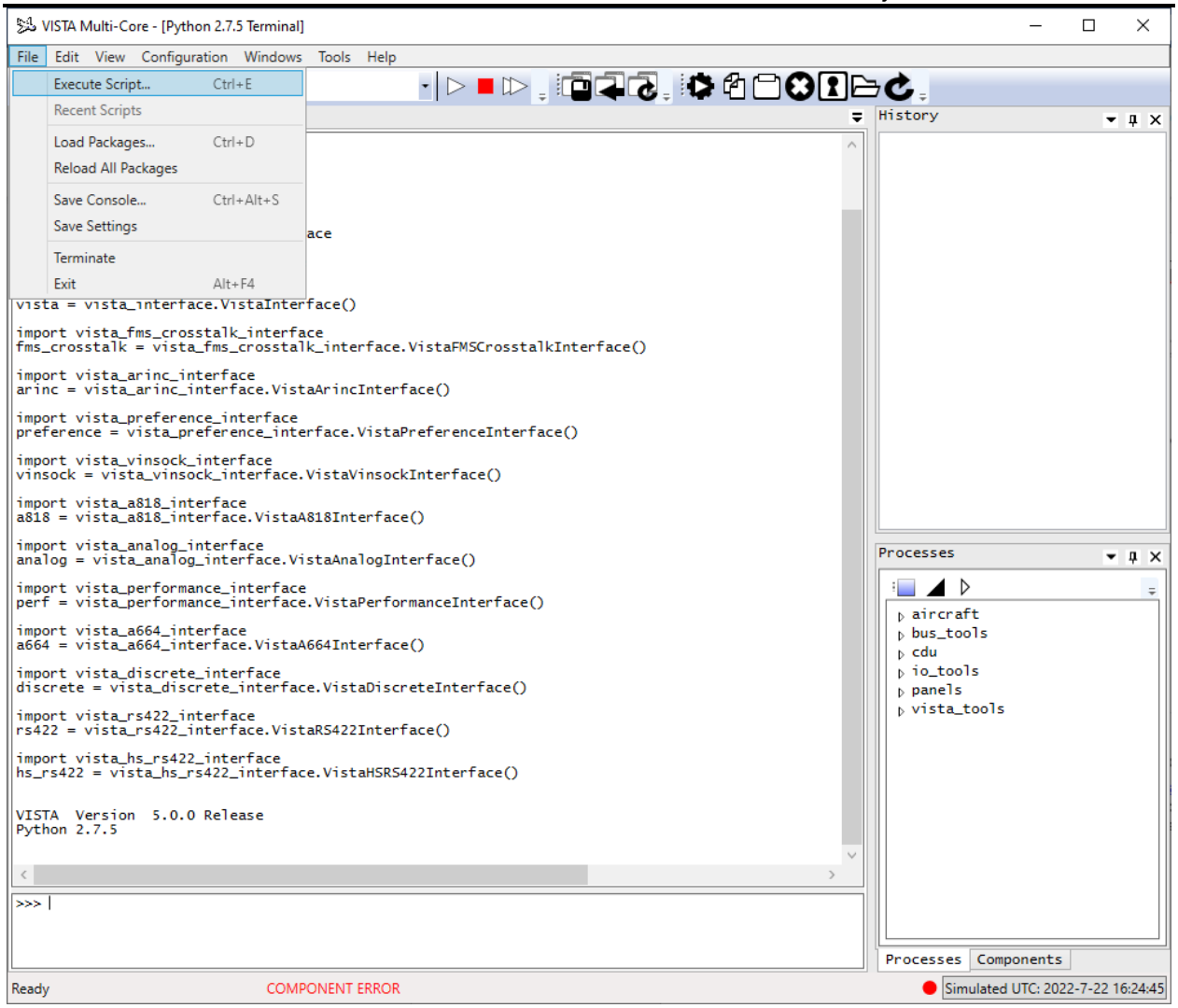
b. <Process_Specific> refers to IPS/EDS/IPS DUAL/EDS DUAL and folders like vista_sim, vista_sim_EDS, vista_sim_EDSDual, vista_sim_IPSDual or vista_sim_MRJ

   Example for IPS is shown in the below figure:



8.1.3.2 VISTA Startup Configuration

Once the vista is launched, test script can be executed from File menu, Execute script.



A startup file

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/utilities/util_DLCA_Startup.py has been created to create all the Vista processes. Enter the required options for 'Enter Aircraft Type:', 'Enter Environment Type:', 'Enter Simulation Type:' after startup file is chosen for execution. Calling this Start-up file after VISTA is launched creates all the specified Processes. It contains the creation of required processes like AFD, Analyzer, DLCA, PM, Data Manager, etc, which is shown below:

create('a_host_pm')

create('l_dlca')

create('atc')

create('dlcagen')

create('agps')

create('dm')

create('dlcatracetool')

create('chimetester')

create('analyzer')

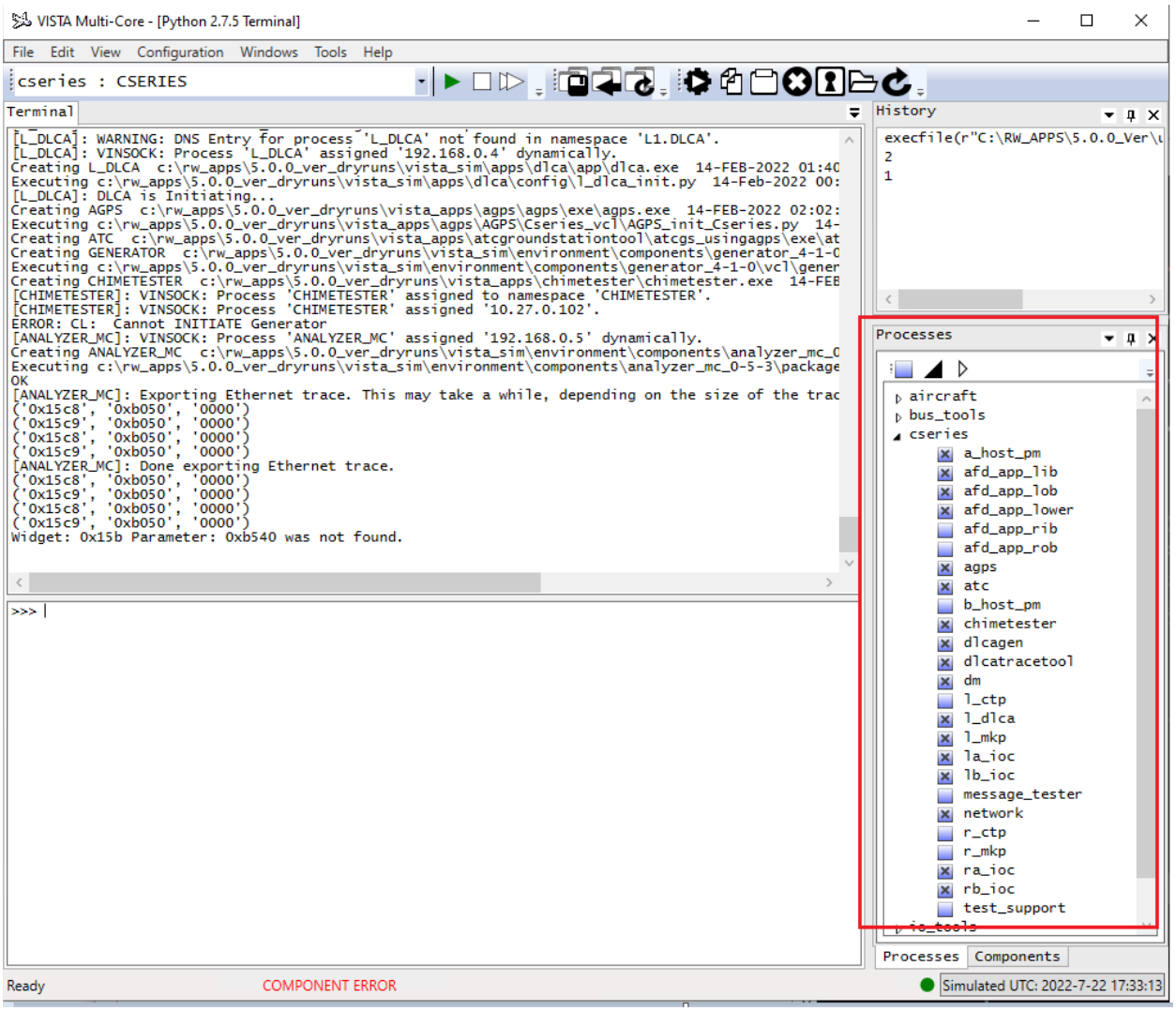
create('l_ccp')

create('afd_app_left')

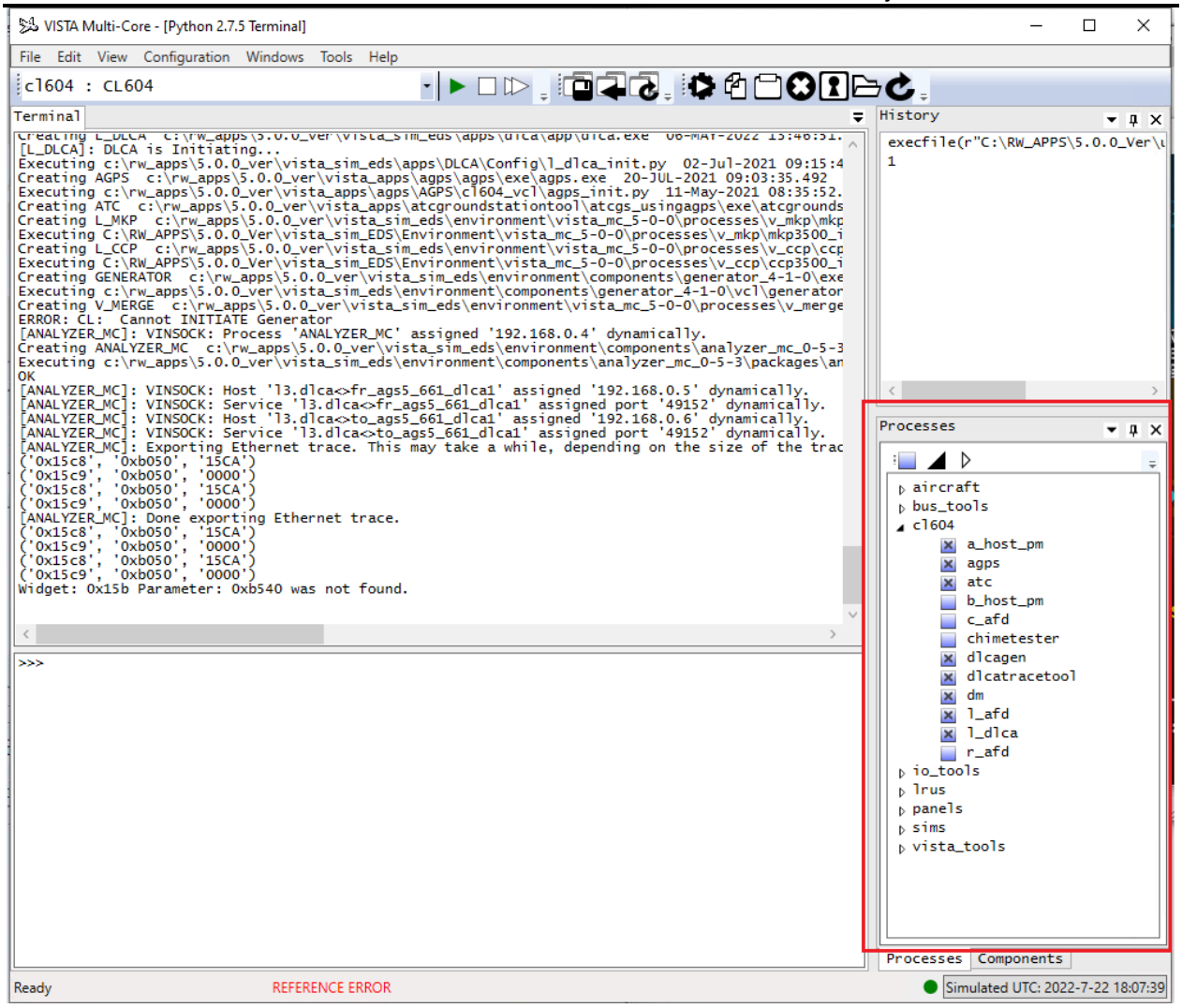
Calling this Start-up file after VISTA is launched creates all the specified Processes.

Refer the below figures for Process tab references in EDS and IPS environment.

IPS Environment:



EDS Environment:



8.1.3.3 Terminating the VISTA Session

VISTA session can be ended at any point of time by performing any of the following:

QUIT: This will always terminate the VISTA session

CTRL-Y: This will abort all VISTA processes. It is not the recommended way to end a session, because no process shutdown/finalizing code will be executed. The command "vista.kill('*')" can be used to kill all the Vista Processes (or) Kill_Vista_MC.exe *(Path:* https://asvn/csdlnkver-dlca-a661/branches/<X.X.X_Ver>/<Process_Specific>*)* can be used to kill all the Vista MC and all Vista Processes.

Note: Type vista.<Function_name> to perform multiple operations in vista tool like vista to wait for specified time, kill the process, create the process…etc.

Similarly for AGPS, ATC Ground Station, DLCA Data Generator, DM Tester, Trace tool

AGPS → agps.<Function_name>

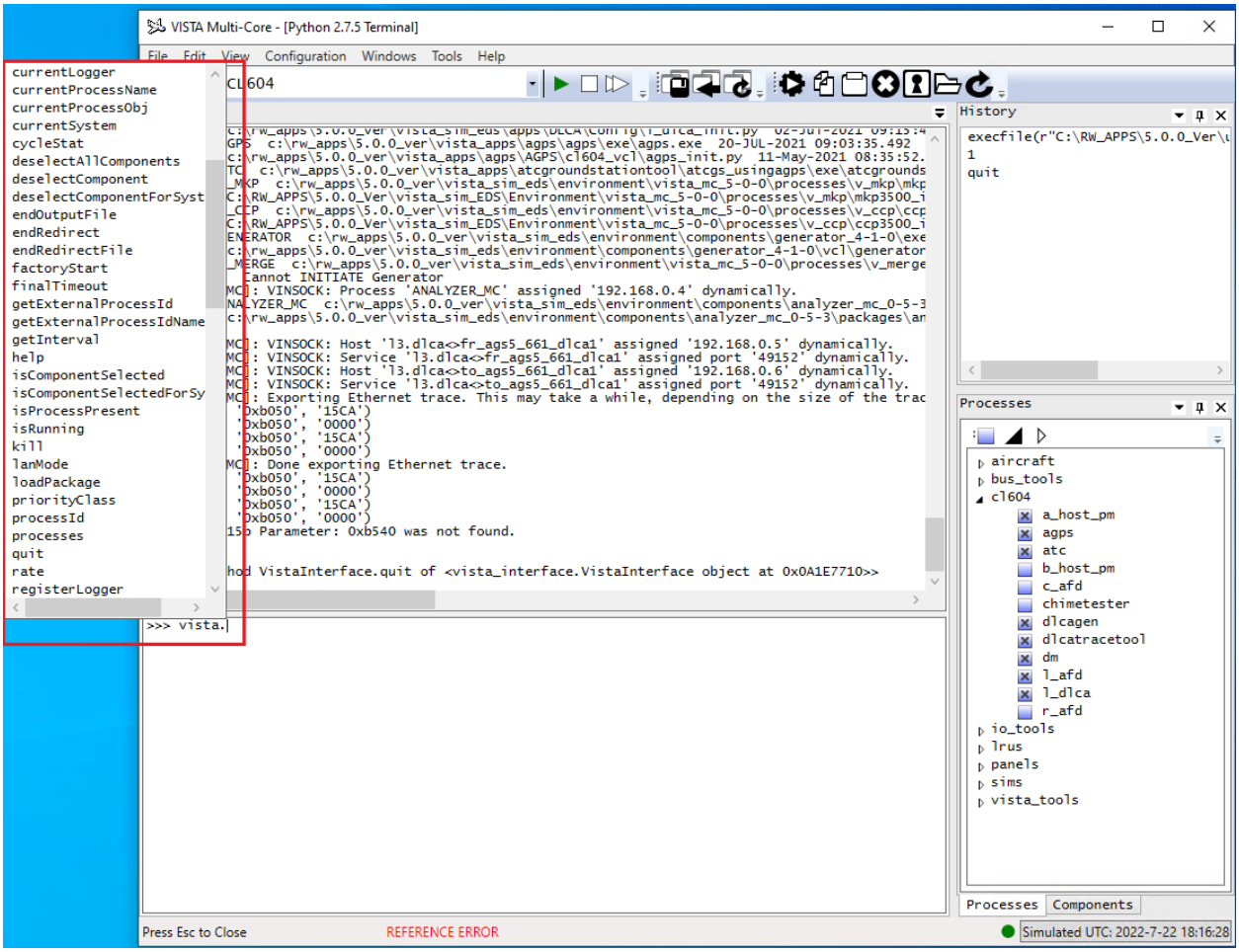
ATC → atc.<Function_name>

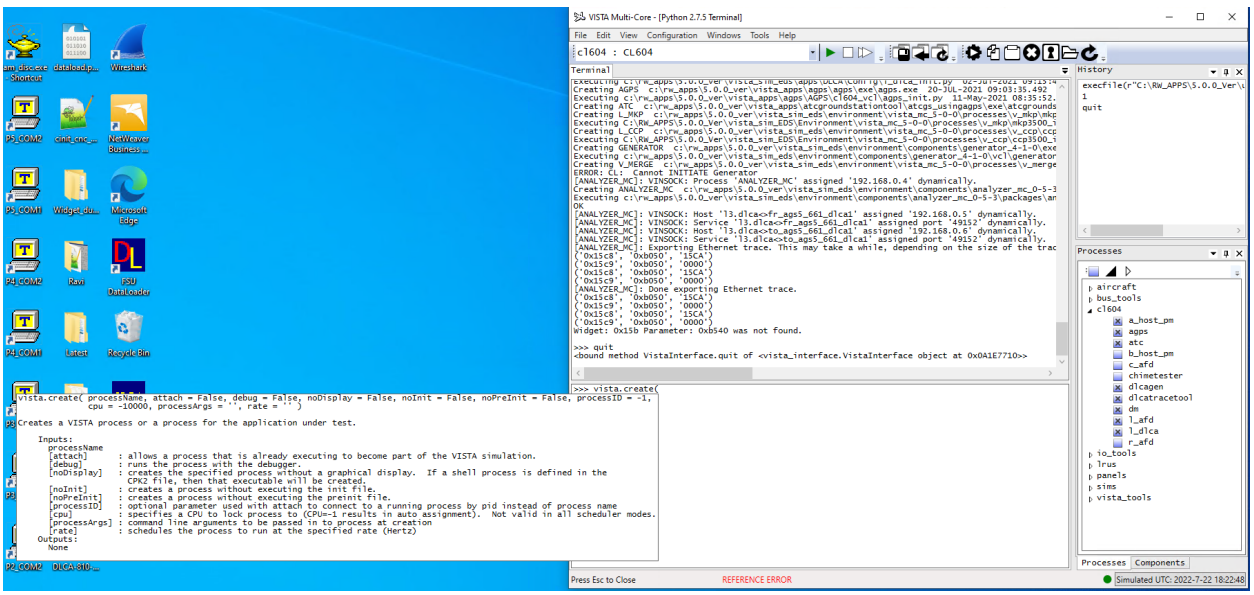
DLCA Data Generator → dlcagen.<Function_name>

DM Tester → dm.<Function_name>

Trace tool → dlcatracetool. <Function_name>

Refer below figures for commands and parameters:





8.1.4 VISTA Interactive Operating Mode

When you run VISTA interactively, you can enter commands at the Python prompt or create and use processes through the graphical interface. VISTA also supports a command line recall feature that "remembers" the preceding commands. You can navigate through the stored commands using the up and down arrow keys.

8.1.5 Interrupting Script Execution

You can temporarily halt execution of a running script file using the key sequence CTRL-SHIFT-C. This will force a PAUSE command, which suspends execution of the script and brings up a PAUSE> prompt.

Once at the PAUSE> prompt, you can use the CONTINUE command to resume the script, the STEP command to proceed through the script one command at a time, and the CANCEL PAUSE command to exit the script file and return to the normal Python prompt.

8.1.6 VISTA Processes

The DLCA tools created in VISTA are: AFD, CCP, MKP, ATC Ground Station, DLCA Data generator, AGPS, Data Manager Tester, DLCA trace tool.

8.1.6.1 AFD (Adaptive Flight Display)

This is a Pilot viewable display device. Using MKP and CCP, navigation to different pages and responding /creating messages can be done.

EDS:

IPS:

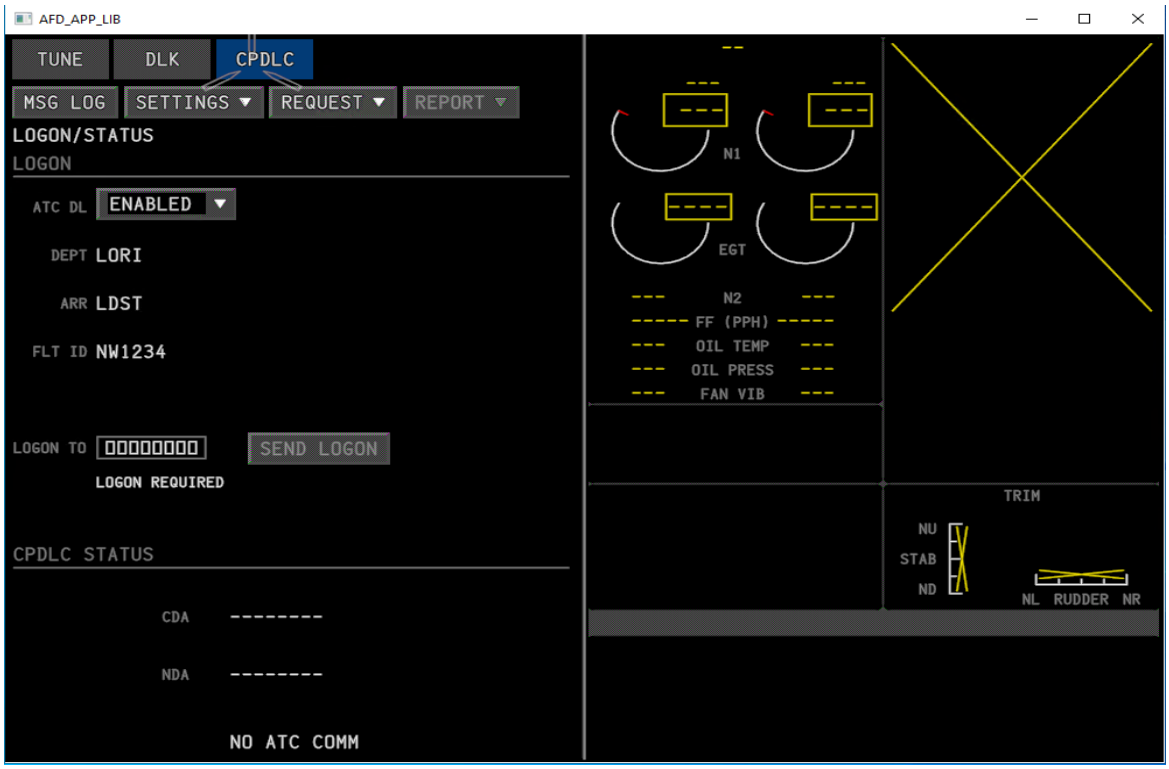


Figure 18: Example AFD display

8.1.6.2 CCP (Cursor Control Panel)

The CCP simulation allows interaction with the AFD displays.

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_MKP_and_CCP.ppt

EDS:

IPS:

Figure 19: Example CCP

For the automated CCP commands, refer the Vision Users guide Section 4 for VisionLib.Click(). Vision supports the below selections using the Click command:

["upr"|"lwr"|"menu"|"castleup"|"castledown"|"castleleft"|"castleright"|"pushselect"|"ptt"]

Example Commands:

VL.Click("castleleft ",100, "l")
VL.Click("menu",100, "l")

8.1.6.3 MKP (Multi-Functional Keyboard Panel)

MKP simulates a Multi-Functional Keyboard Panel. In addition to accepting script commands to control interaction with the simulation, a graphical display of a MKP faceplate is also provided. The keys on the MKP graphical interface can be used as they would be on an actual MKP, and the screen also accurately reflects the state of the MKP display.

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_MKP_and_CCP.ppt

EDS:

IPS:

Figure 20: Example MKP

For the automated MKP commands, refer the Vision Users guide Section 4 for VisionLib. PushButton_MKP(). Vision supports the below MKP selections for the PushButton commands:

['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '.', '/', ' ', '{ENTER}','{+/-}', '{CLR_DEL}', '{MEM}', '{CHART}', '{FMS}',

'{DIRECT_TO}', '{SYS}', '{CNS}', '{CAS}', '{PREV}', '{NEXT}', '{CNCL}', '{EXEC}', '{UP}', '{DOWN}', '{LEFT}', '{RIGHT}']

Example commands:

VL.PushButton_MKP("1","l")

VL.PushButton_MKP("{CLR_DEL}","l")

8.1.6.4 ATC Ground Station

ATC Ground Station is a ground station application which provides means to send CPDLC, AFN, CM and ADS messages to pilot and view the downlink messages sent by the pilot to ground station. ATC Ground Station works in two different environments. It can be used to connect with AGPS or an AAR interface.

In the Message Log window of ATC Ground Station, all ground station uplinks and all aircraft downlinks are displayed. Hex representation of each uplink message or downlink message can be viewed in Raw Data tab of ATC Ground Station. ATC Ground Station is automated / scripted for VCL/Python commands.

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_ATC.ppt

EDS/IPS:

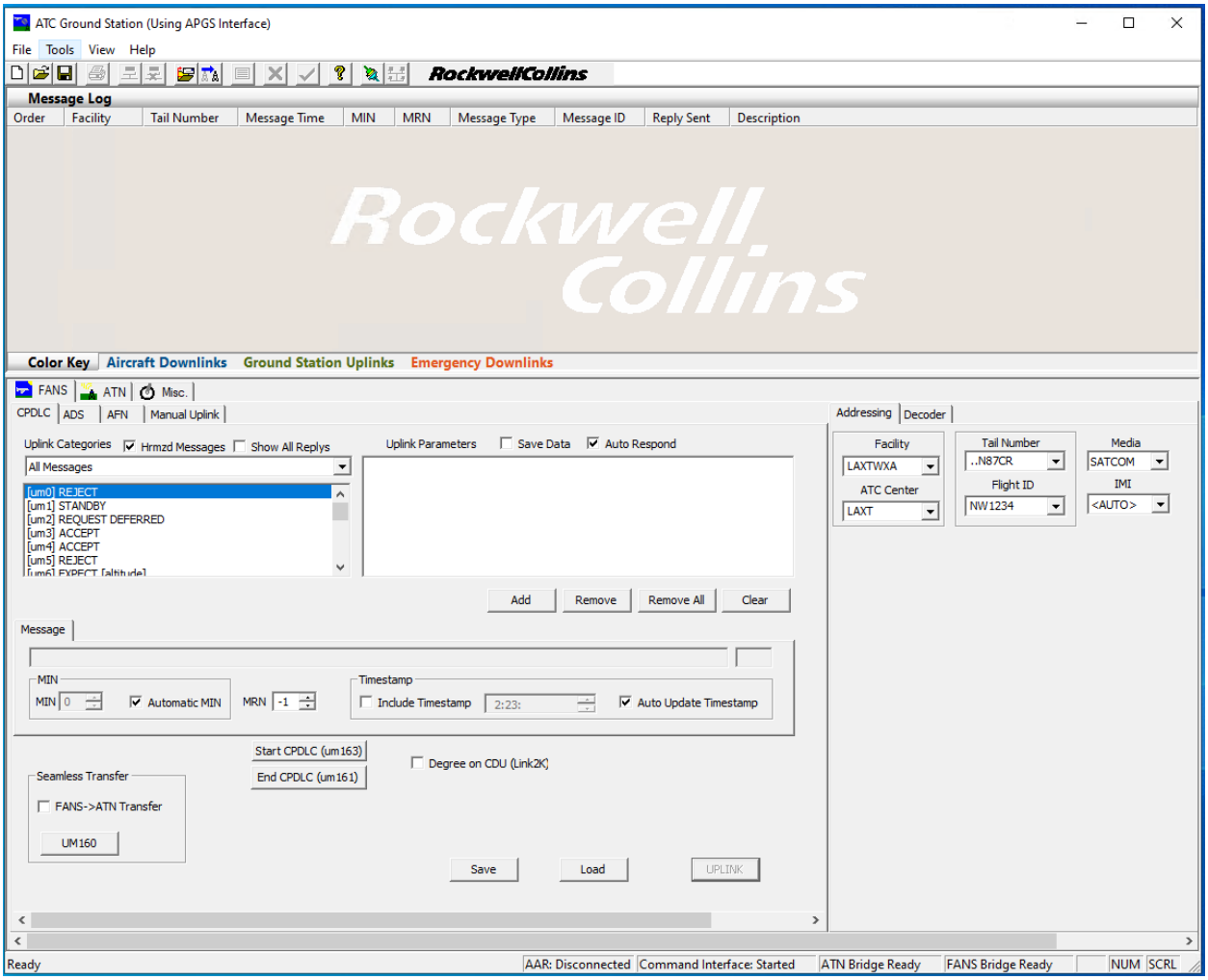


Figure 21: ATC Ground Station

In Message Log window of ATC Ground Station, all ground station uplinks are displayed in green color and all aircraft downlinks are displayed in blue color. Hex representation of each uplink message or downlink message can be viewed in Raw Data tab of the ATC Ground Station. Other tabs like CPDLC, ADS Uplinks, AOC, AFN, Free Text and Message Text are available in Data link Tester for uplinking messages.

Under CPDLC tab, when "Show Advanced Properties" checkbox is enabled, under Message tab unique MIN number is displayed for every uplink message and MRN should be -1.

This tool is automated to perform the uplinks and return the requested messages to Vista. The automated commands can be found in the file:

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/ATCGroundStationTool/ATCGS_UsingAGPS/test_interface_notes/PythonCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/ATCGroundStationTool/ATCGS_UsingAGPS/test_interface_notes/VCLcommands.txt

Automated command examples:

vista.create("atc")

vista.kill("atc")

atc.set("facilitydesignation EDYY")

atc.set("facilityname ABCD")

atc.set("latencyvalue 200")

8.1.6.5 DLCA Data Generator

DLCAGen provides Flight Id, Tail Number, strapping, License keys and ICAO Address to DLCA. This tool is automated to perform the set and get from Vista.

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_DLCAGEN.ppt

EDS/IPS:

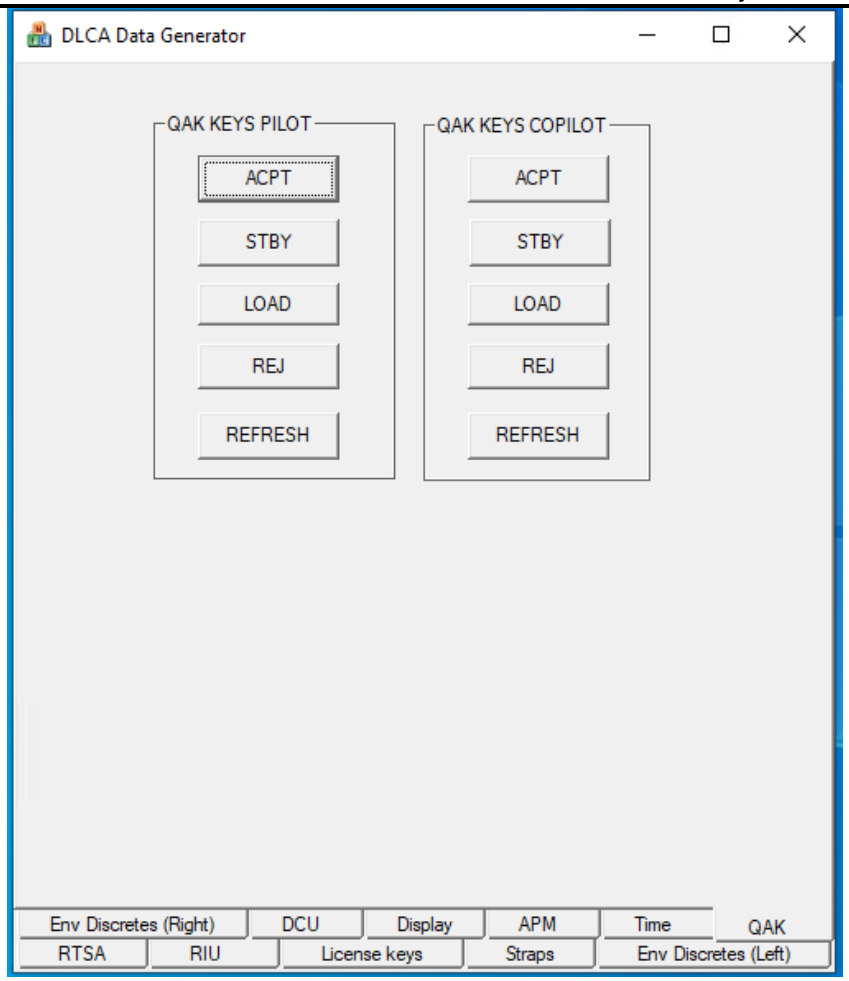


Figure 22: DLCA Data Generator

The automated commands can be found in the file:

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/DLCAGen_DualLicense/exe/PythonCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/DLCAGen_DualLicense/exe/VclCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/DLCAGen_SingleLicense/exe/PythonCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/DLCAGen_SingleLicense/exe/VclCommands.txt

Automated command examples:

vista.create("dlcagen")

vista.kill("dlcagen")

dlcagen.rtsa("rtsa_1 flight_id 1234")

dlcagen.rtsa("rtsa_1 enable true")

dlcagen.rtsa("rtsa arinc_429 true")

8.1.6.6 AGPS (Air/Ground Protocol Stack)

Air/Ground Protocol Stack(AGPS) routes Uplink and Downlink messages between the DLCA and the ATCGroundStation simulator. It also performs the functions of the Printer and Ground Network Message Log Windows provided in the ABLE environment. AGPS sends the broadcast status to report data link availability and which MU is active among other status indications. The AGPS also handles the DSI handshaking with the DLCA and dialogue session management normally performed by the CMU/RIU. AGPS is automated/scripted for VCL/Python commands.

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_AGPS.ppt

EDS/IPS:

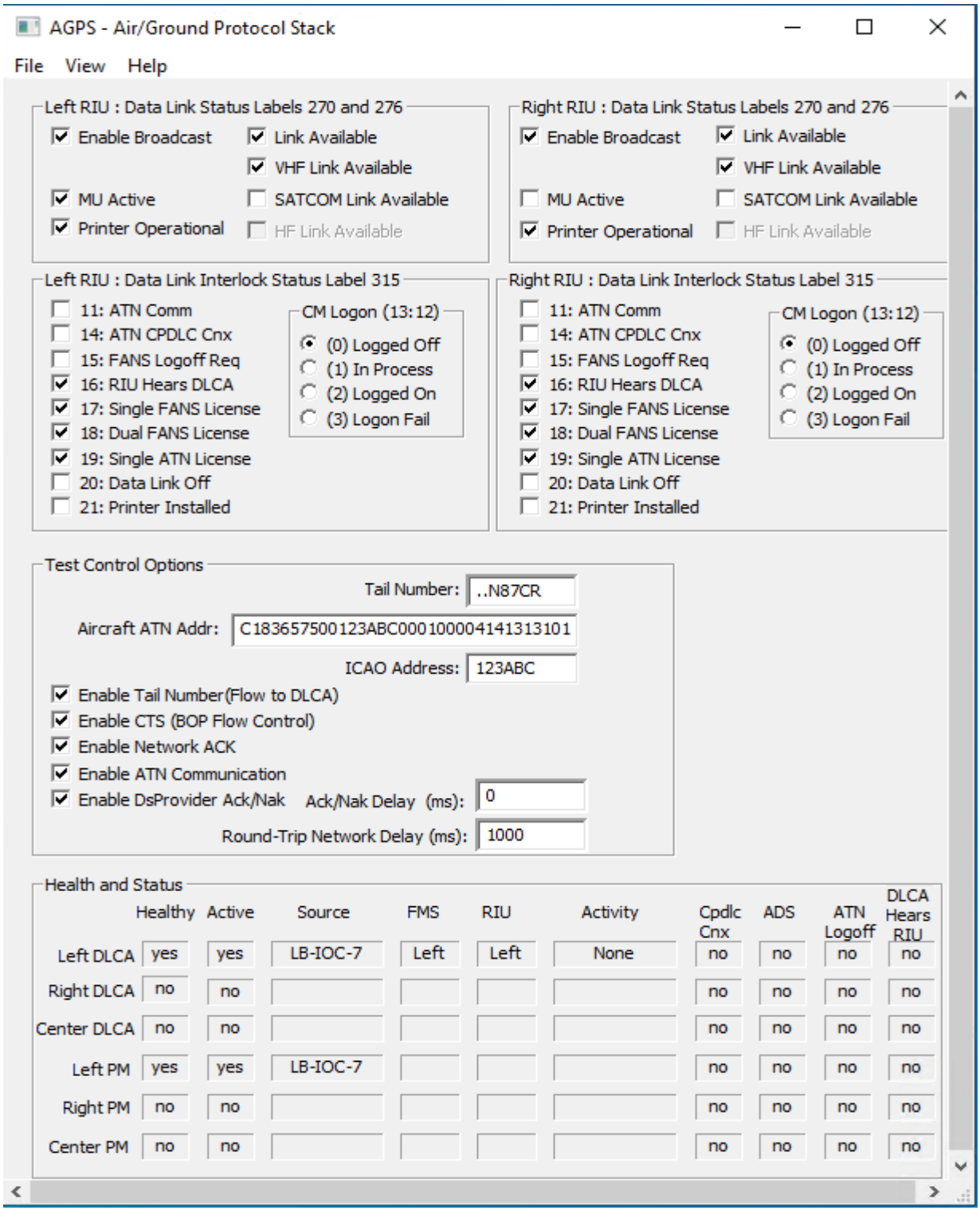


Figure 23: AGPS

The automated commands used for DLCA-6500 verification can be found in the file:

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/AGPS/AGPS/test_interface_notes/PythonCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/AGPS/AGPS/test_interface_notes/VCLcommands.txt

Automated command examples:

vista.create('agps')

vista.kill('agps')

agps.set_mu_active(side='left', state='true')

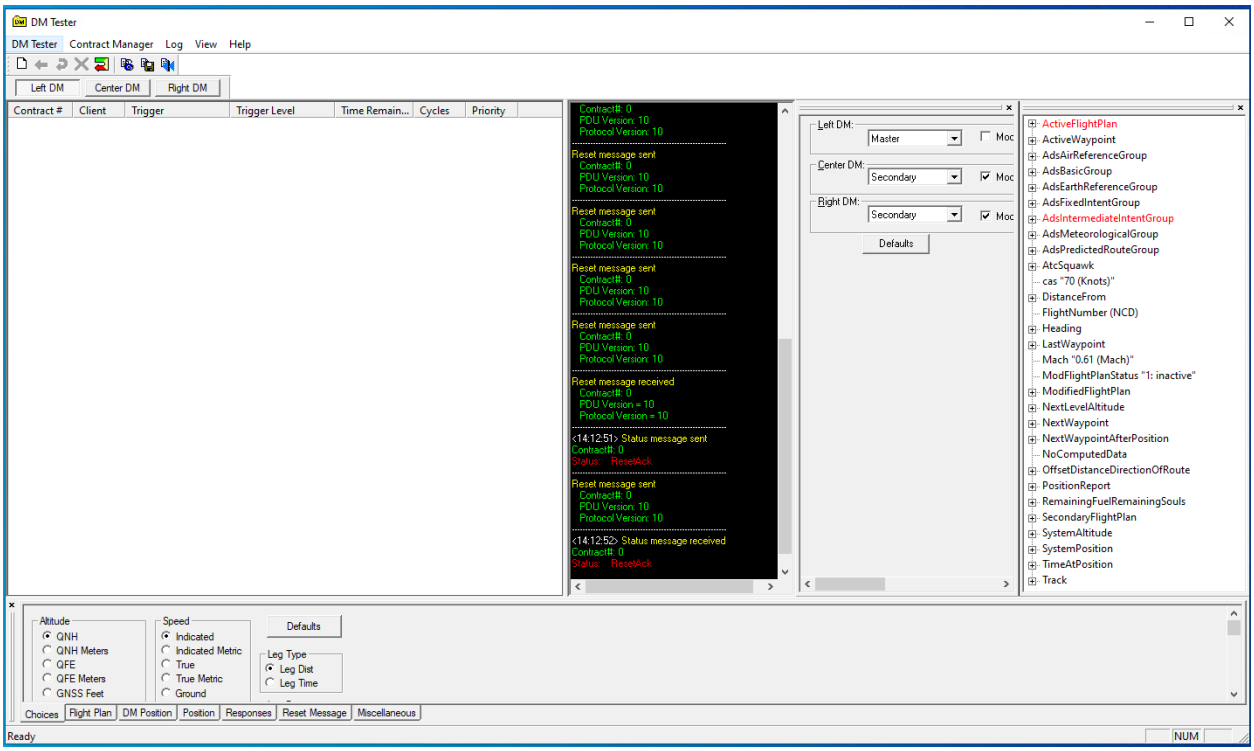
agps.set_printer_available(side='left', state='false')

agps.set_link_available(side='right', state='true')

8.1.6.7 Data Manager Tester

The purpose of the Data Management (DM) Application is to provide a mechanism to transfer information between the DLCA and the FMC.

EDS/IPS:



The DM provides a "client-server" relationship such that the client can load data (in the form of input messages) or request data (in the form of request messages) from the server. The server responds with output and/or status messages. The DM application hosted in the DLCA will act as a client, while the DM application hosted in the FMF will act as the server.

A contract identifier has been introduced in the Data Management Protocol Data Unit (DMPDU), which serves to correlate output or status messages to request messages, and to distinguish simultaneous conversations.

A priority field has been introduced in the Data Management Protocol Data Unit (DMPDU), which serves to identify the relative priority of the DM contract as it relates to the client application. The priority field is defined as an integer ranging from 0 to 31, with 0 having the highest priority and 31 having the lowest.

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_DMT.ppt

The Data Management message set consists of six types of messages, each having different characteristics:

a. Reset messages:

- Following a connection establishment with the underlying data link layer, the DM will send a reset message to the peer DM.
- Resets may be initiated by either the client or server. All reset messages will contain a contract number of zero and a priority of zero (highest).
- Resets may be initiated by either the client or server. All reset messages will contain a contract number of zero and a priority of zero (highest).
- The expected response to a received reset message will be a resetAck status message with contract number of zero and a priority of zero. In this context, this status message is referred to as the reset acknowledge message.
- The resetNak status message is referred to as the reset negative acknowledge message.

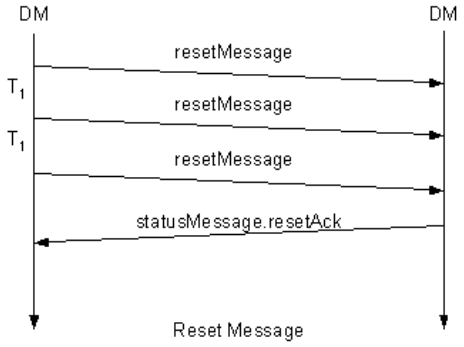


Figure 24: Reset messages

b. Input messages

There are two types of Input messages Validate and Load, Interactive Validate and Load.

Below diagram explains the Validate and Load messages:

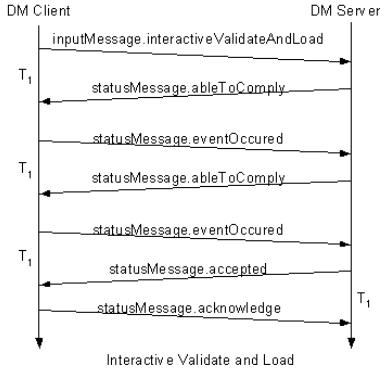


Figure 25: Validate and Load messages

Below diagram explains the Interactive Validate and Load messages:

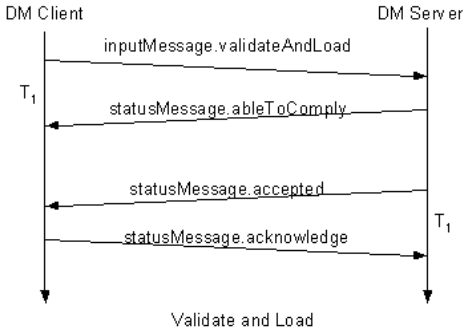


Figure 26: Interactive Validate and Load messages

c. Request messages

There are three types of Request Messages: Periodic, Event, Immediate.

Below diagram explains the Periodic Request messages:

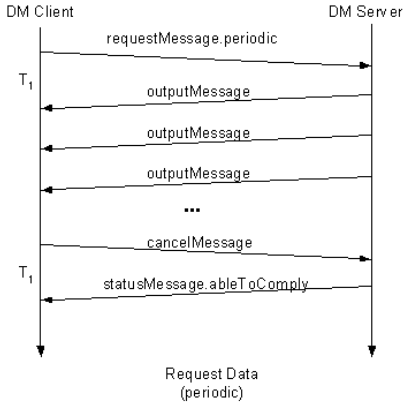


Figure 27: Periodic Request messages

Below diagram explains the Event Request messages:

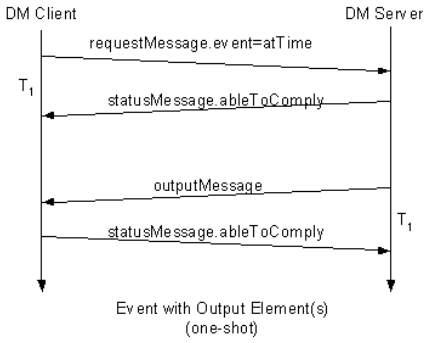


Figure 28: Event Request messages

Below diagram explains the Immediate Request messages:

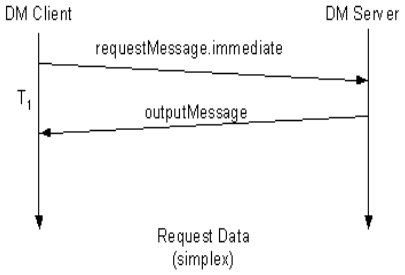


Figure 29: Immediate Request messages

d. Output messages:

- The server generates an output message to the client with the message fields set to the following values outputelem-seqOf = <sequence of output elements containing requested data>
- If any of the output elements are invalid, the server will substitute the output element "noComputedData" in place of the normal output element in the output message. When the server responds with a "noComputedData" output element, it means the server recognizes and has access to the element but that it is currently invalid

e. Status messages: This gives different types of message Status like AbleToComply, UnableToComply, UnableToProcess, UnableToDecode, Not Ready, Accepted, Rejected, Standby, Partial Load, NoLongerAbleToComply, EventOccurred, MaxMsgSizeExceeded, StatusAcknowledge, Aborted, InProgress, TimedOut, ServerFault, ResetAck, ResetNak

f. Cancel messages:

Below diagram explains the Cancel Messages

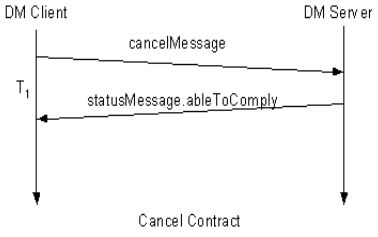


Figure 30: Cancel messages

Data Manager also acts as an FMS which supplies the current aircraft information like Altitude, Speed, Heading, etc. These values can be set and fetched from DM Tester and used to create the Request message/Status downlink message.

This tool is automated to fetch set the values in DM Tester (FMS) and fetch the log values printed in DM Tester. The automated commands can be found in the file:

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/DMTester/PythonCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/DMTester/VclCommands.txt

Automated command examples:

vista.create("dm")

vista.kill("dm")

dm.master("left master")

dm.master("left secondary")

dm.master("left mod_flight_plan_active false")

dm.reset_server("left")

8.1.6.8 DLCA Trace Tool

Trace Tool is used to print the debug information of DLCA when the respective bits are enabled. The bit mapping is present in the SRS for DLCA-6500 HMI, section System Info Page.

EDS/IPS:

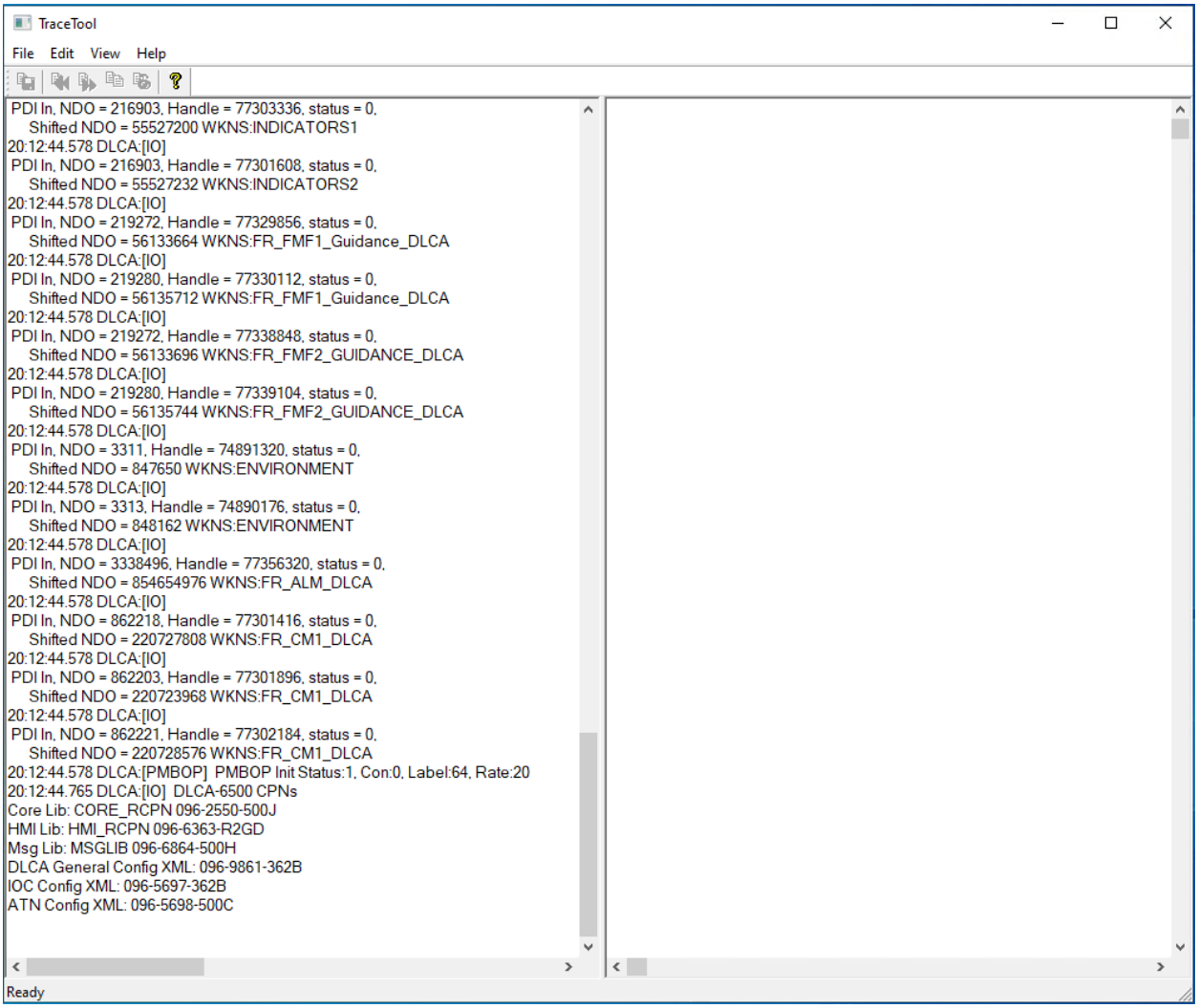


Figure 31: Trace Tool

This tool is automated to fetch the debug information getting printed on the log window and also set the filter for individual debug bit. The automated commands used for DLCA-6500 verification can be found in the file:

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/TraceTool/PythonCommands.txt

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_apps/TraceTool/VCLCommands.txt

Automated command examples:

vista.create("dlcatracetool")

vista.kill("dlcatracetool")

dlcatracetool.log("LEFT CLEAR")

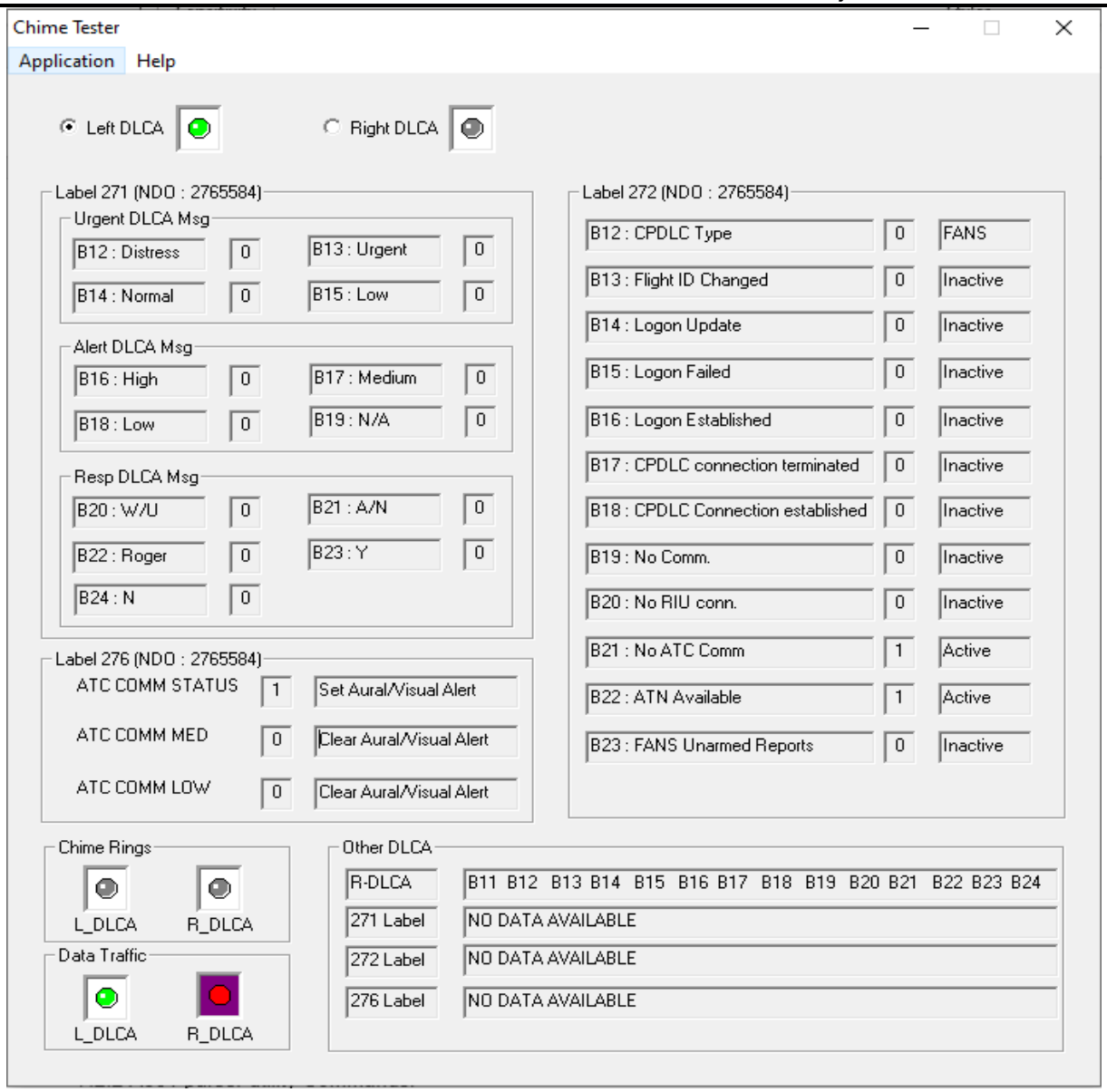
dlcatracetool.log("LEFT GET_LINE_COUNT")

Refer to the located path for more information about this tool:

https://asvn/csdlnkver-dlca-a661/documents/Training/DLCA6500_TRACETOOL.ppt

8.1.6.9 Chime Tester

Chime Tester tool is the User Interface to display the bit status of DLCA to EICAS NDOs as shown in below figure in IPS/EDS environments.



8.2 VISTA Test Tool

8.2.1 A661 parser utility Framework:

DLCA Test Framework is an automated testing environment based on A661 Parser utility functionality. The interaction between AFD-x and DLCA is Arinc-661 protocol based. DLCA and AFD-x interact with each other based on events. For example, Button click by the pilot on AFD is an event. Whenever the event happens, DLCA updates status of widget's properties based on the software requirements. The entire trace of communication between the AFD-x, DLCA and the vice-versa is continuously captured by the VISTA_Analyzer. A661 Parser utility saves this trace as temp_trace_buffer.atb file locally.

While running the Test Procedure, the test statement will request the A661 Parser utility to return the Properties of a particular widget. A661 parser utility considers the request and searches the

temp_trace_buffer.atb and returns back the latest state of that particular widget. Find the following figure to understand the architecture of A661 framework.

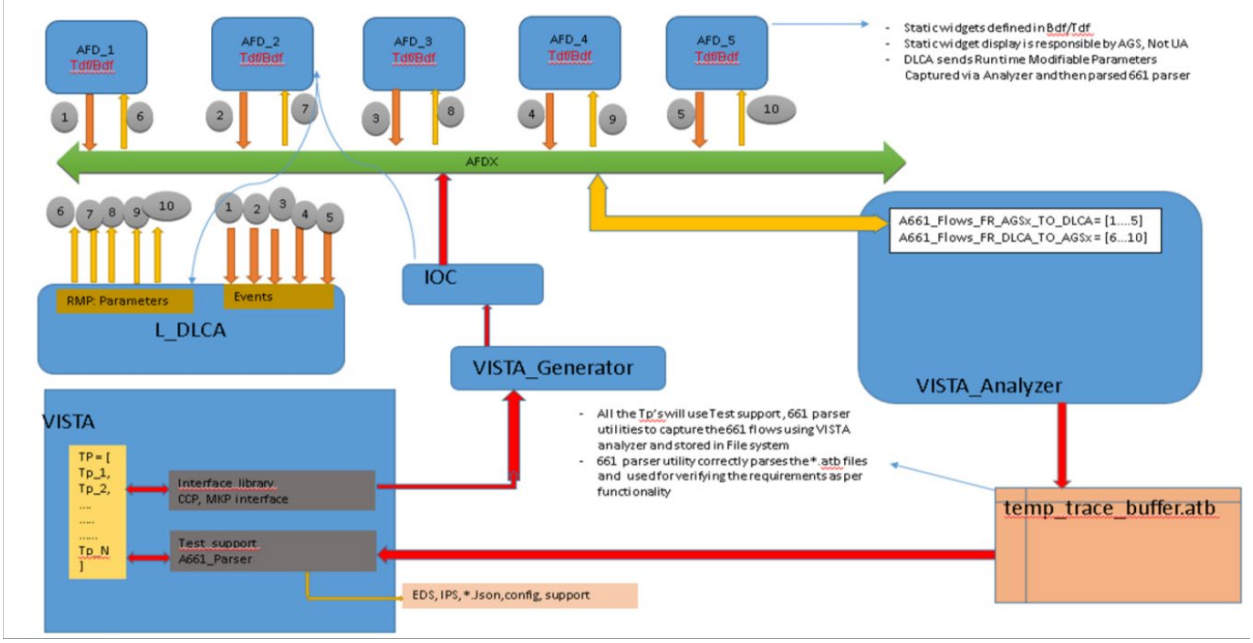


Figure: A661 Framework

8.2.2 A661 parser utility Commands:

The Test procedure use the Test Support Utilities which will call the A661 Parser utility commands.

Most commonly used A661 Parser utility commands are:

openTrace()	Clears the previous trace. Opens the file provided at "file_loc"
convertA661String()	Converts 661 block into string of text.
parseA661Commands()	Parses an A661 command into a dictionary containing the appropriate sub-element.
findA661Commands()	Iterates through an A661 block of data and passes all command headers to the parser.
printA661Trace()	Prints the latest log to a file at C:\temp and creates a CSV to include the parsed data at the same location.
find()	Performs a lookup of the most recent event matching the parameters of the provided.
findAll()	Performs a lookup of the most recent event matching the parameters of the provided.
findColorCodes()	Replaces the color code with a user readable block.
findSizeCodes()	Replaces the size code with a user readable block.
clearValues()	Clears data from the most recent trace

8.2.3 Test Support Utilities

To use the A661 Parser Utility commands more efficiently in the test script, some utilities have been written and are placed in the below path:

https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/utilities/util_Test_Support.py

https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/utilities/A661_Parser_2.py

Below are some of the utility commands created to improve scripting:

- util_Test_Support. findWidgetEvent (widget, Parameter): This function will use the provided data by looking at the most recent A661 Value for the given Widget and RMP combination
- util_Test_Support. selectItemByIndex():This function is used to select the choice from selectlist for which the list of choices already defined in the utility from the requirement.
- util_Test_Support. performLogon():This function is used to Logon To Ground using the flight id and logon to fields.
- util_Test_Support.enterTextByWidget(): This function is used to enter the text to editable entry boxes.
- util_Test_Support.selectWidget(): This function is used to move the cursor to widget location and select that widget.
- util_Test_Support.gotoPage(): This function is used to goto specific page from any other page.

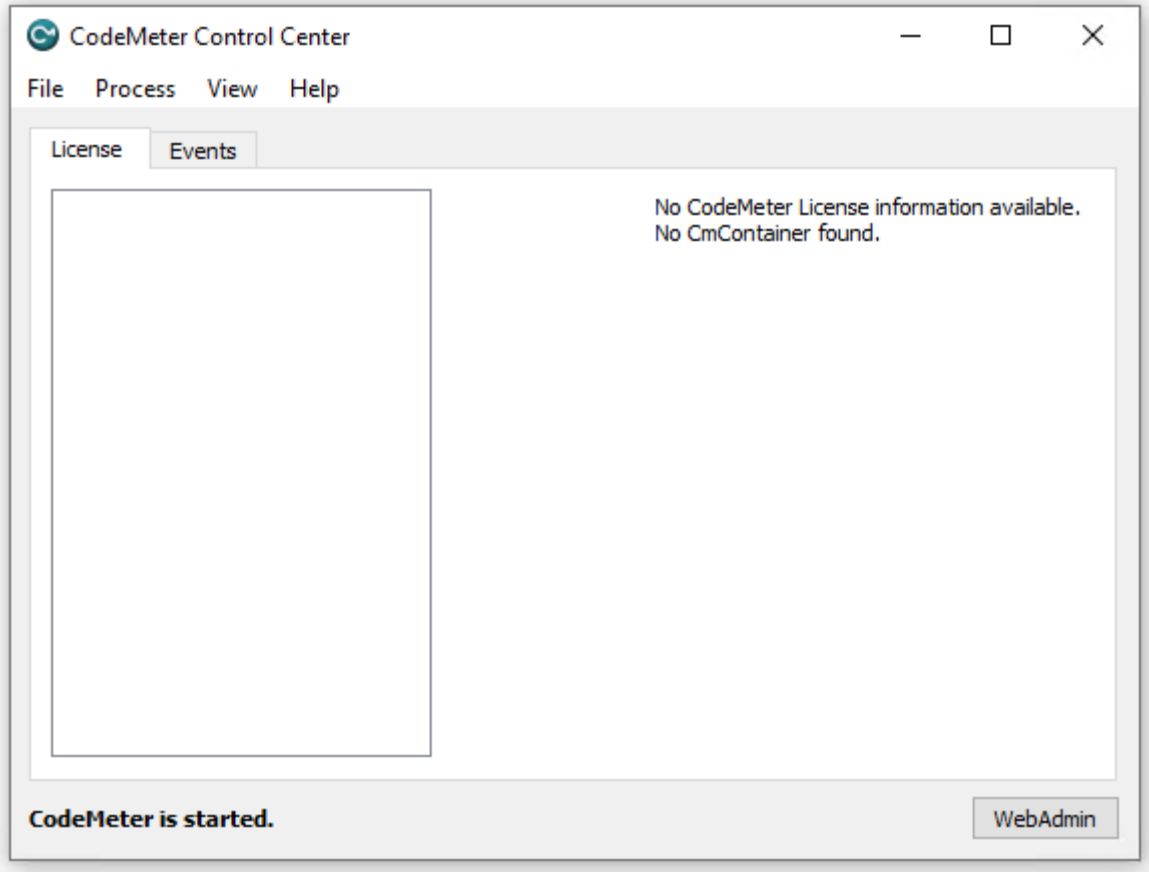
8.2.4 CodeMeter and diamond license for vista

CCANET Domain:

1. Install CodeMeter from the path <Dev_branch>/Simulation/EDS_VistaSim_Dual/Tools/Diamond/CodeMeterRuntime.exe, by following the default installation wizard steps.

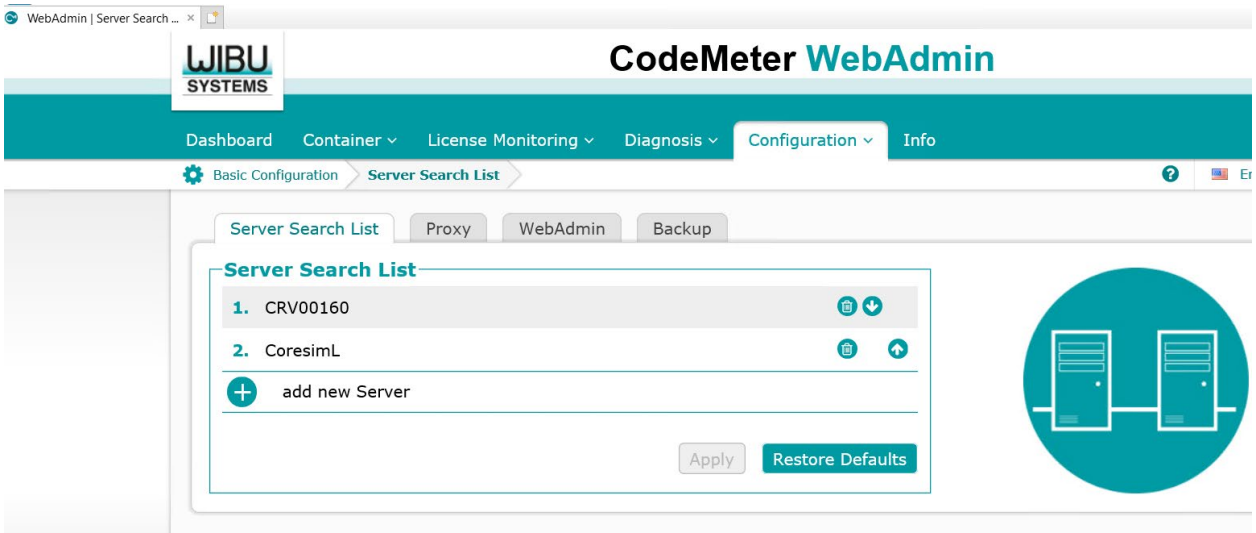
2. Double click on <Dev_branch>/Simulation/EDS_VistaSim_Dual/Tools/Diamond/Diamond_CCANET_License_Server_CRV00160.reg, and select yes in the Registry Editor for adding diamond license server to the Registry.

3. To add License server to the CodeMeter server search list, open CodeMeter Control Center, by searching for it in the start menu. Once that is open, click "WebAdmin" in the bottom right.

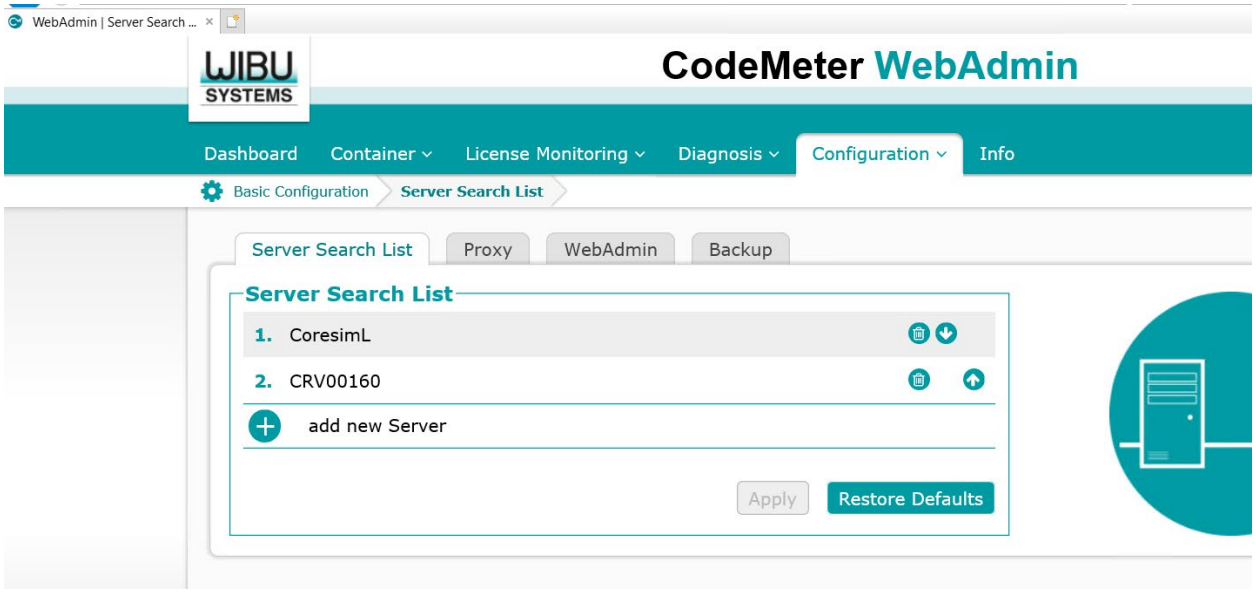


4. Click on Configuration

5. click "add new Server"



6. Type in "CRV00160" without the quotes, and then click add, then click apply.



Alliance Network Domain:

For Alliance Network Domain ,follow instructions under CCANET Domain to Add the server "CollinsIm01 as shown in above screenshots.

NOTE: These steps are applicable for any PC's(Target, Farm, personal PC) where Vista environment has been running.

8.3 Host-based Test Station

8.3.1 Description/Overview

The host-based test station consists of Cockpit and Ground station simulation which are executed from Vista environment in a single PC setup. All the simulated tools and DLCA software are placed in the below SVN path:

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_sim

Or

https://asvn/csdlnkver-dlca-a661/branches/X.X.X_Ver/vista_sim_XXX

8.3.2 Setup

When setting up the workspace for the host-based testing, capture the version label of the software build being tested so that the version information can be recorded with the test results.

8.3.2.1 VISTA PC Instructions

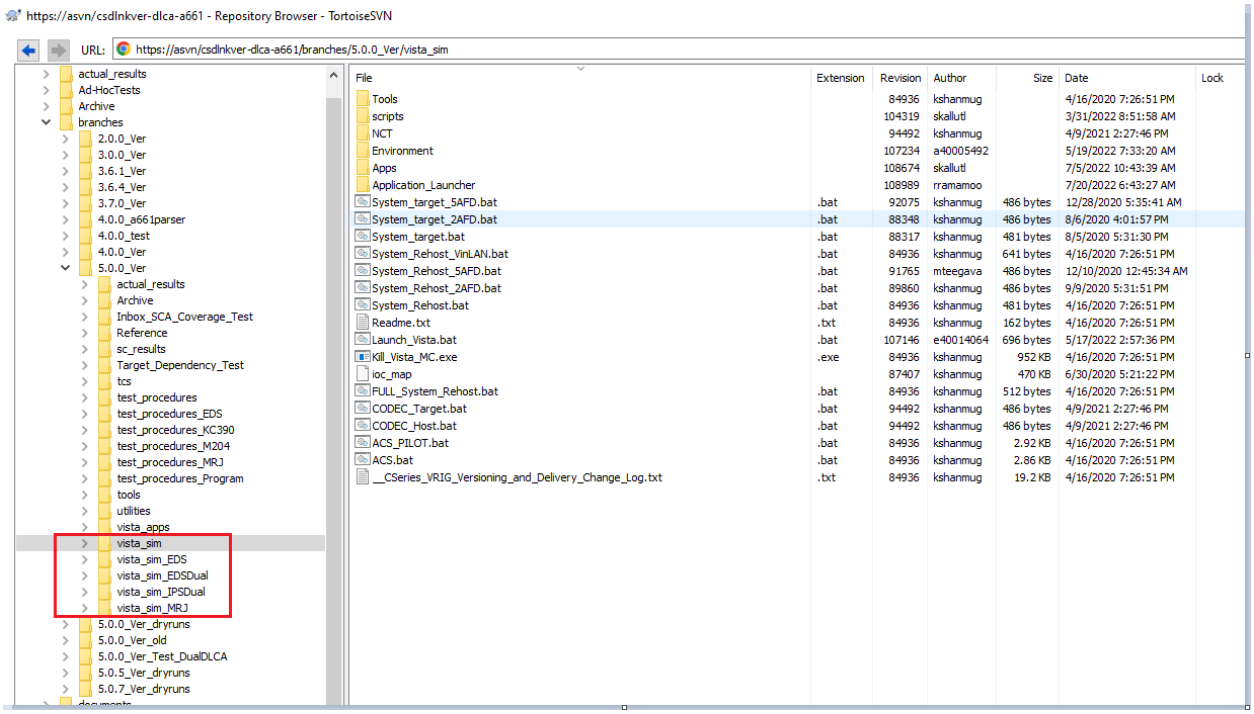


Figure 32: VISTA PC Instructions

Checkout the folder ../vista_sim (or) ../vista_sim_XXX folder(s) from repo as per respective program, execute the batch file which contains 'host' as part of file name like System_Rehost.bat, System_Rehost_2AFD.bat, System_Rehost_5AFD.bat, System_Rehost_VinLAN.bat, dlnkRehost.bat…etc

Double Click on any of the host file in vista_sim/ vista_sim_XXX folder(s), It will execute all the DLCA software and simulated tools such as

- l_dlca, atc, dlcagen, agps, dm,
- dlca tracetool, chimetester, signal analyzer,
- l_ccp, left afd, right afd, a_host_pm, a_mio,
- l_mkp, la_ioc, lb_ioc, ra_ioc, rb_ioc, b_host_pm

Note: Below images are just for reference for ../vista_sim/System_Rehost.bat, it may vary for different programs/products.

Below are some of the examples of all the tools. It may vary from environment to environment.

IPS Environment example:

1. From the L_CCP, select UPR which will show the cursor on the AFD window:

Figure 33: Left Caste from L_CCP

2. From the L_CCP, then select the Menu button, Navigate to AFD window, which will display the below screen.

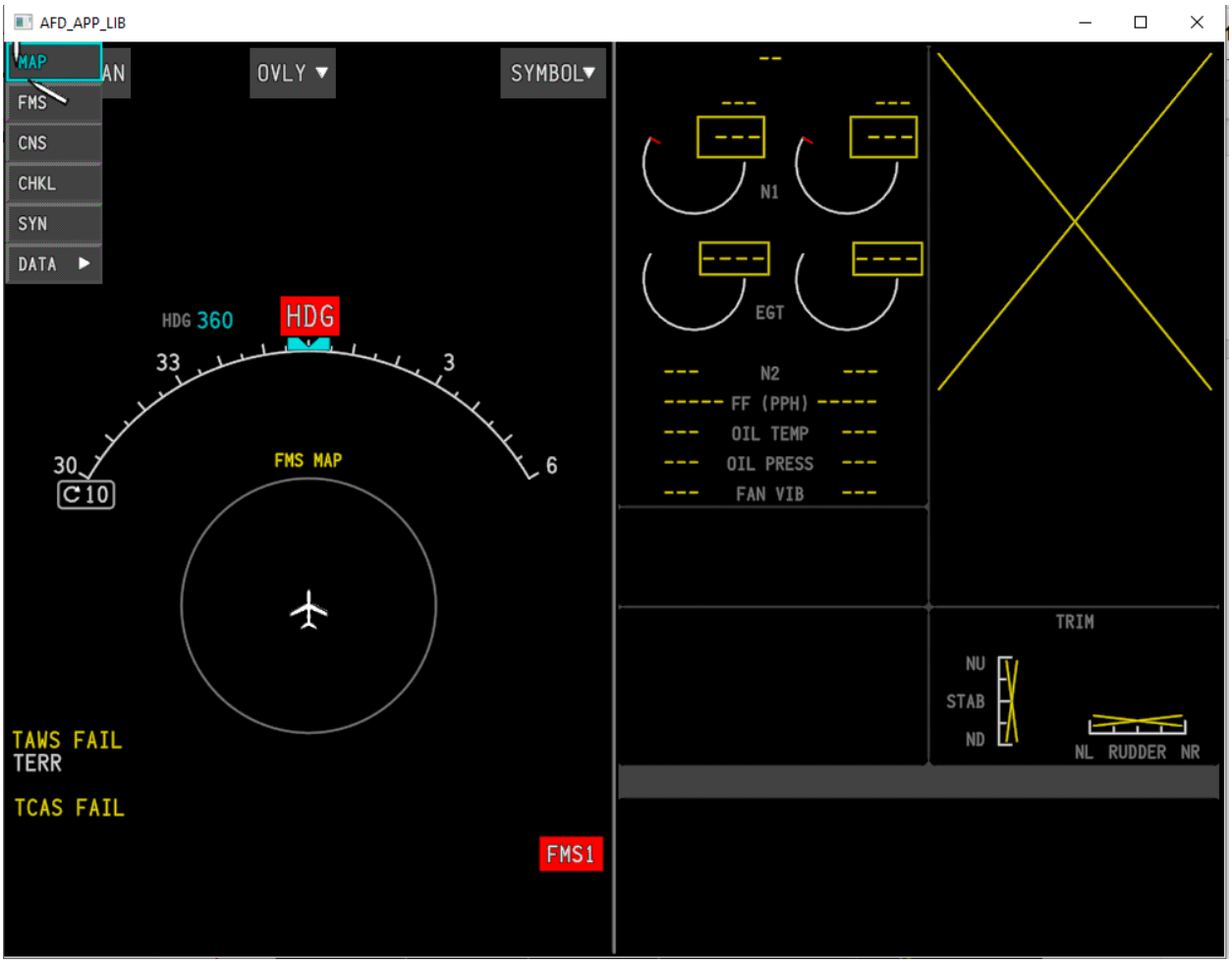


Figure 34: AFD window – Step 1 of 4

3. Using the L_CCP cursor, select the CNS from the menu displayed on the L_AFD which will display the below screen.

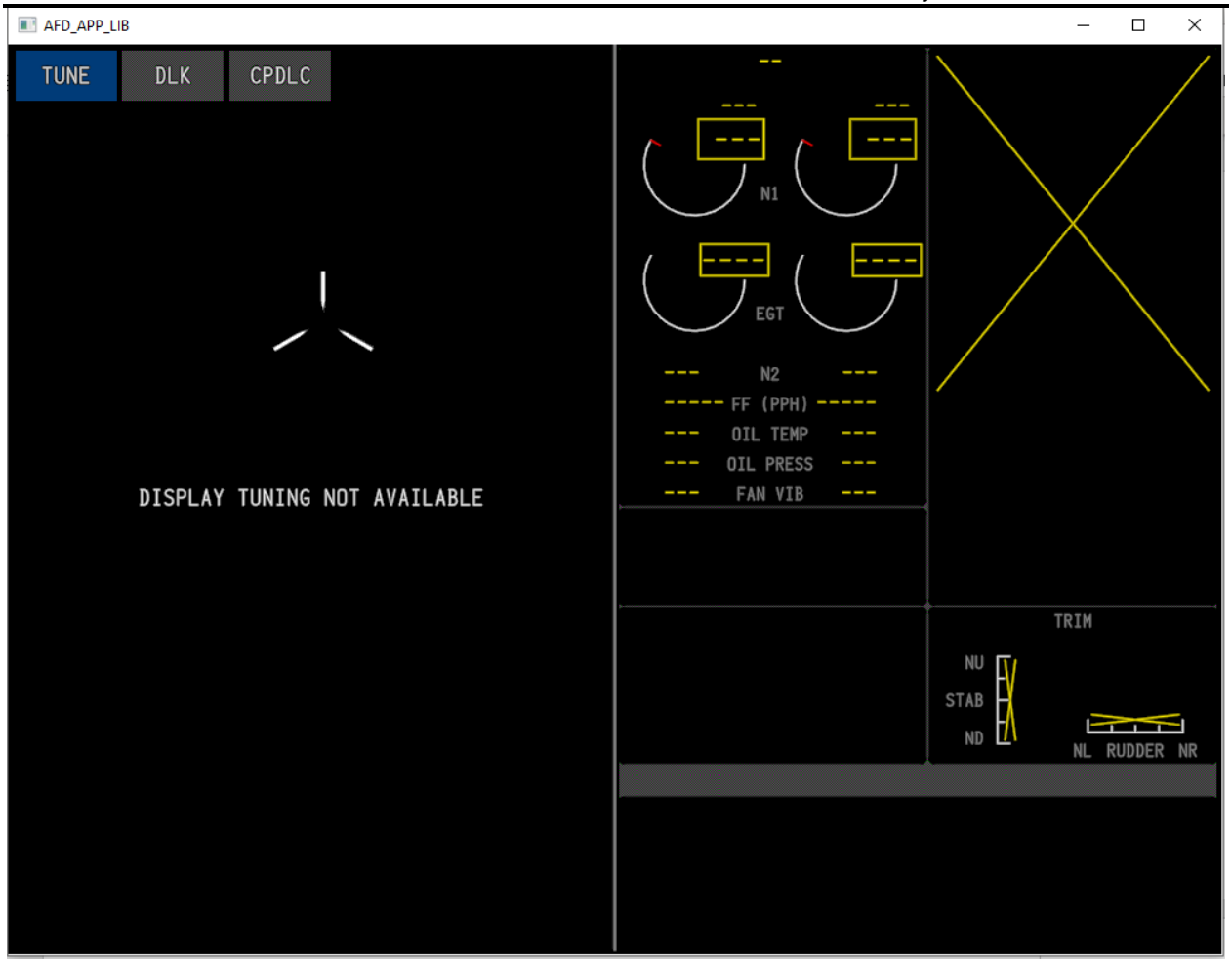


Figure 35: L_AFD window – Step 2 of 4

7. Using the L_CCP cursor, select CPDLC displayed on the L_AFD which will display the below screen.

Now the CPDLC application should be displayed on the L_AFD with LOGON/STATUS page as below:

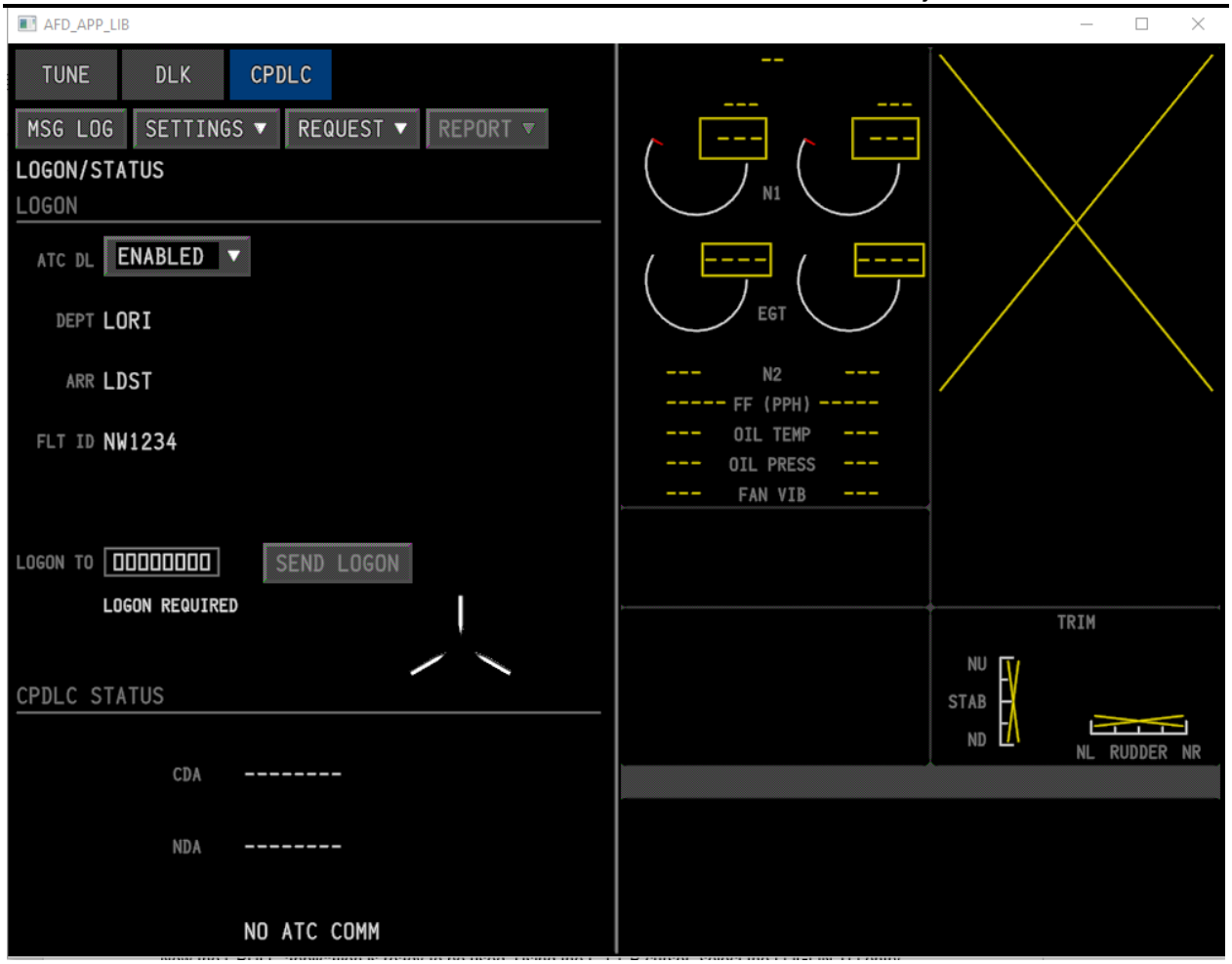


Figure 36: L_AFD window – Step 3 of 4

8. Now the CPDLC application is ready to be used. Using the L_CCP cursor, select the LOGON TO entry field and from L_MKP enter the value "ABCD" and press ENTER button on L_MKP.

Figure 37: LOGON TO entry from L_MKP

9. Then using the L_CCP cursor select the SEND LOGON button on the L_AFD.

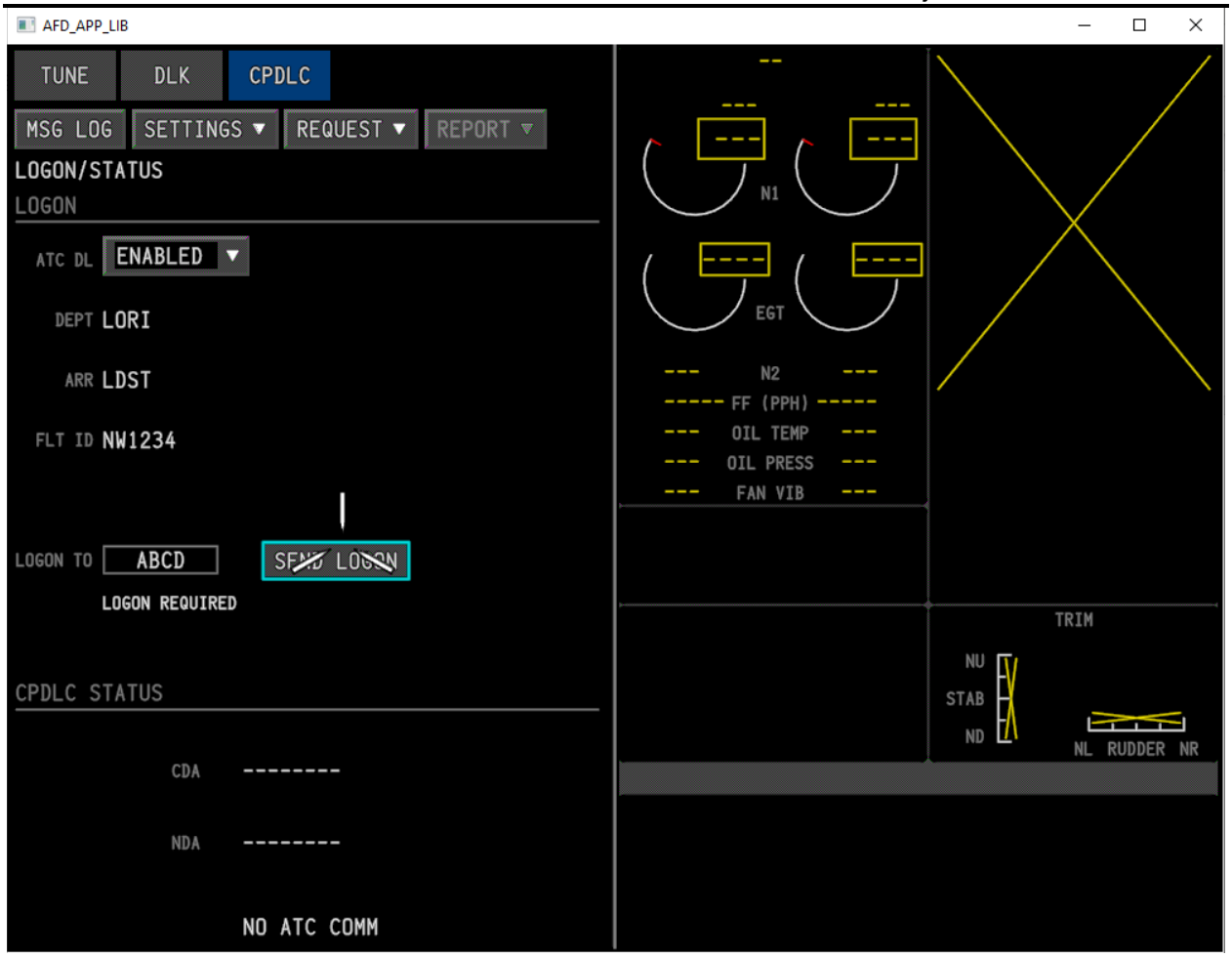


Figure 38: L_MKP window – Step 4 of 4

10. Go to the ATC Ground Station application and you should see the below downlinks which confirms that you are able to perform a logon to the ground:

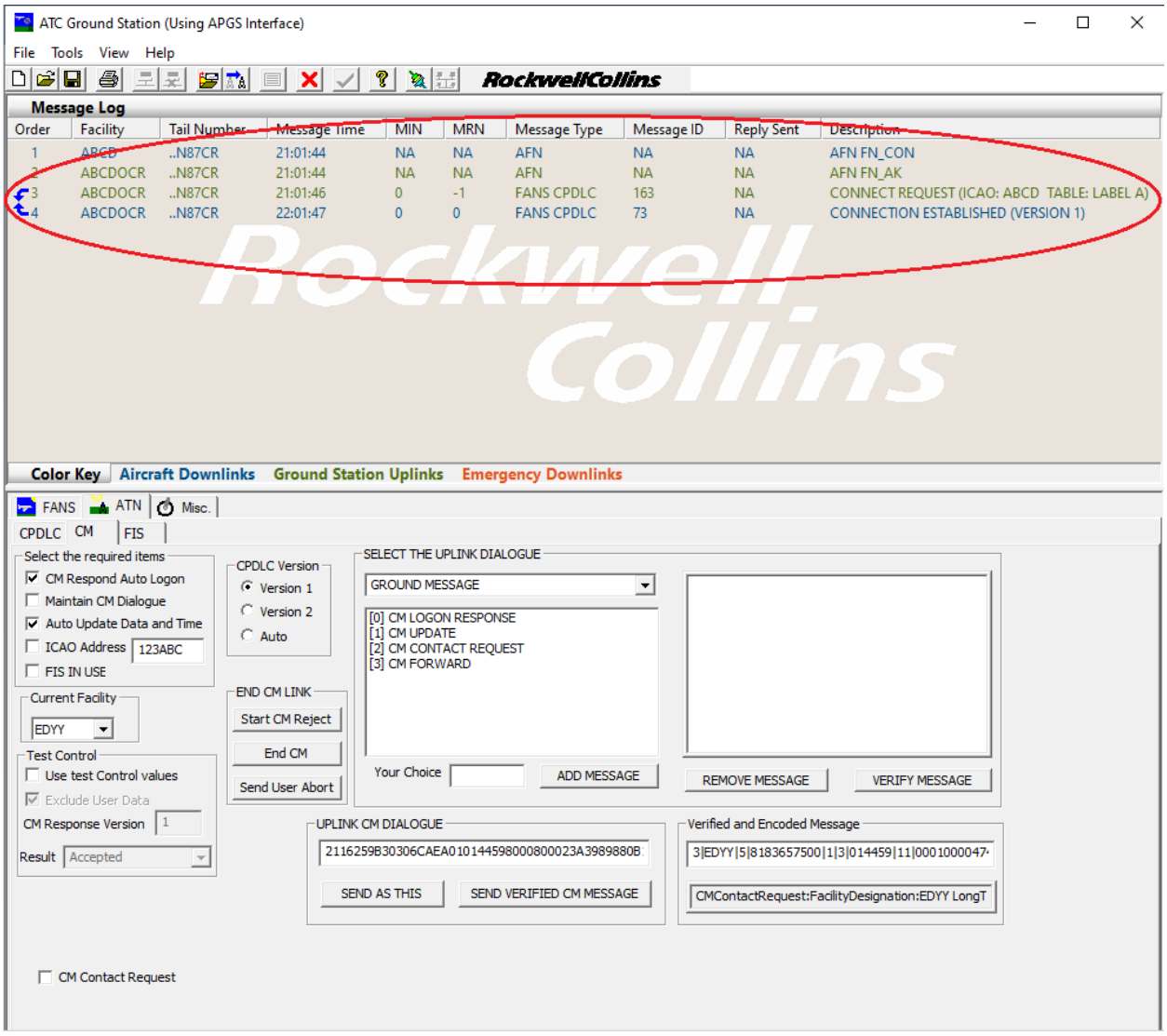


Figure 39: Example of Downlink message displayed in ATC Ground Station

11. Also, the LOGON/STATUS PAGE on L_AFD is displayed with the status as below

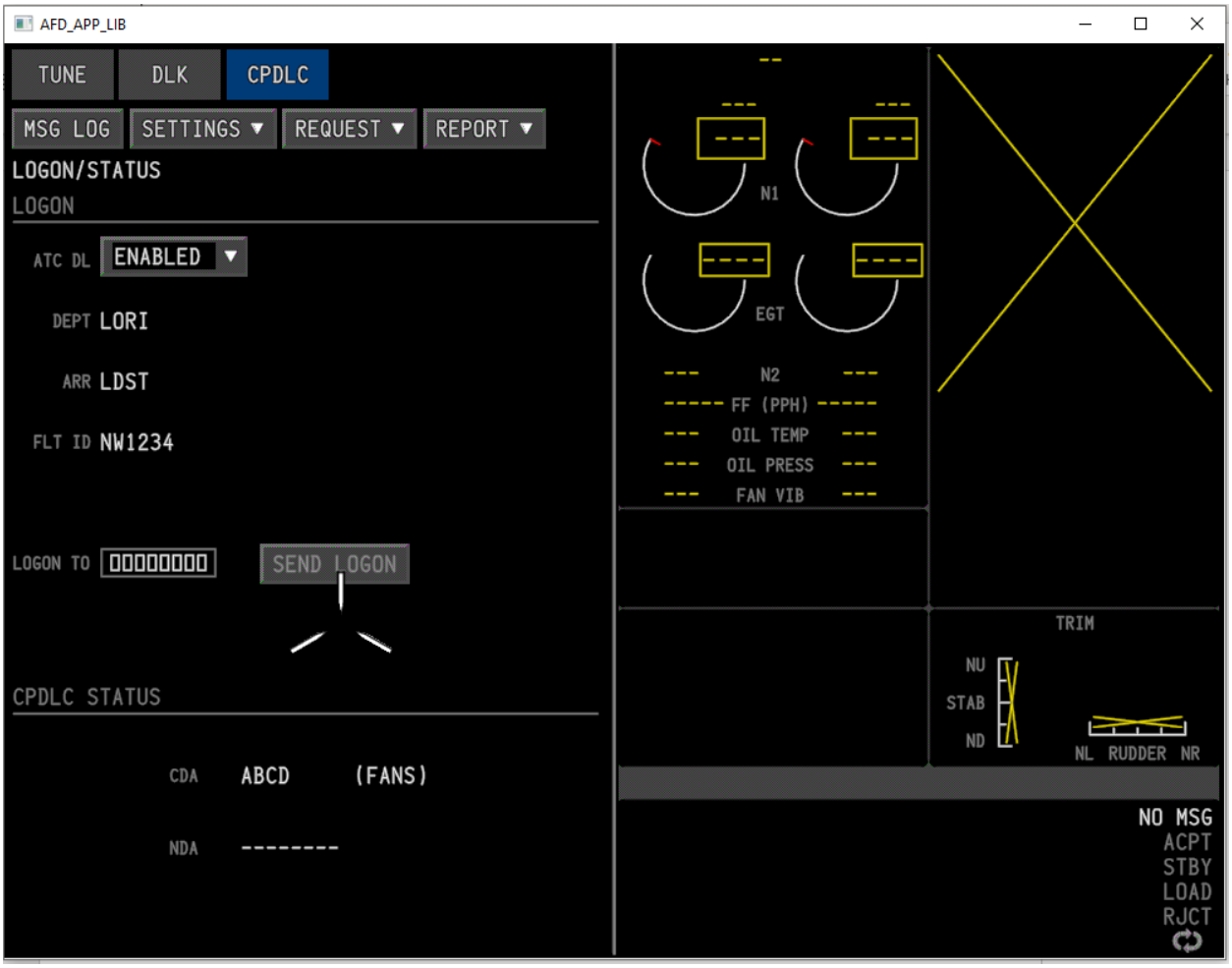


Figure 40: Logon/Status page LOGGED ON TO ABCD

Once the logon is performed, Uplinks/downlinks can be send and tester can perform the required operations.

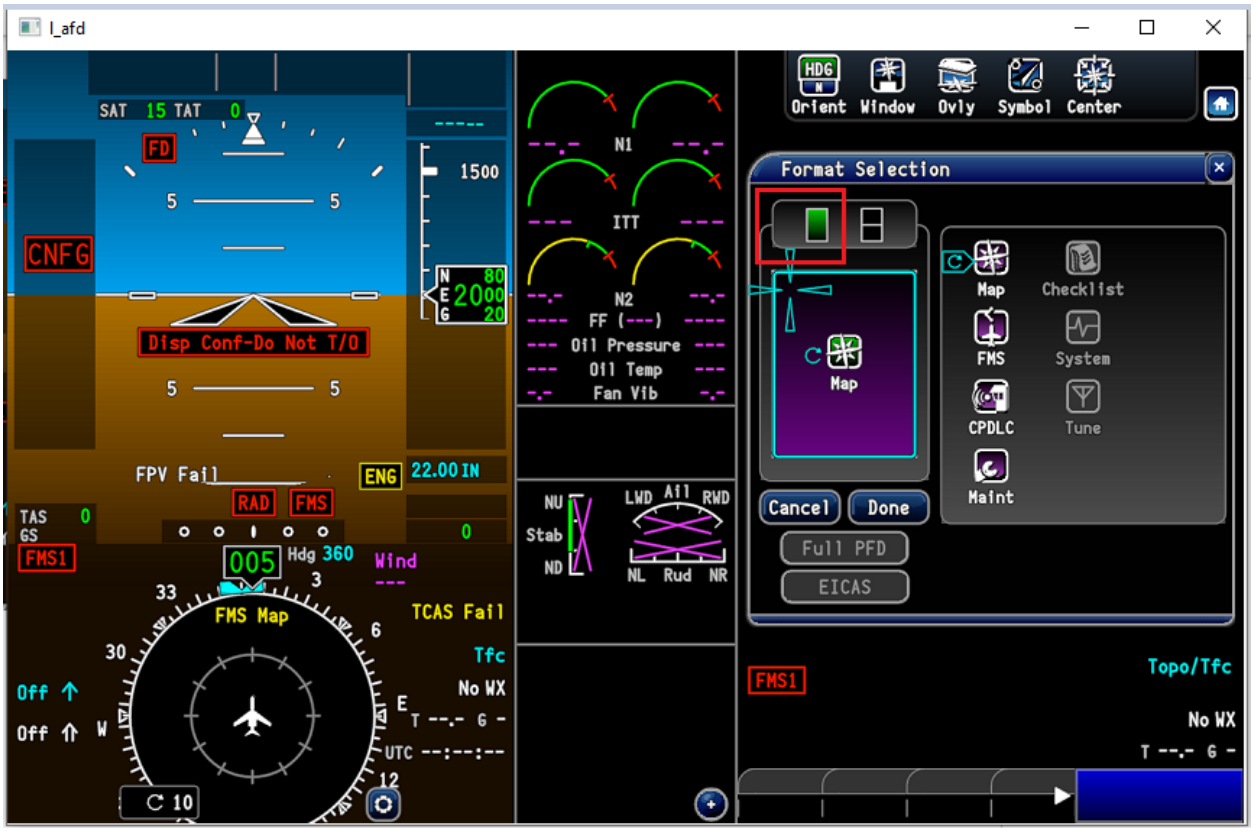
EDS Environment example:

Below images are just for reference for ..\vista_sim_EDS\dlnkRehost.bat, it may vary for different programs/products.

1. In L_AFD, click on the Icon as shown in below figure

2. Click on Home Icon, which will open Format selection as shown in below figure

3. Click on single selection window to enable CPDLC, as shown in below figure

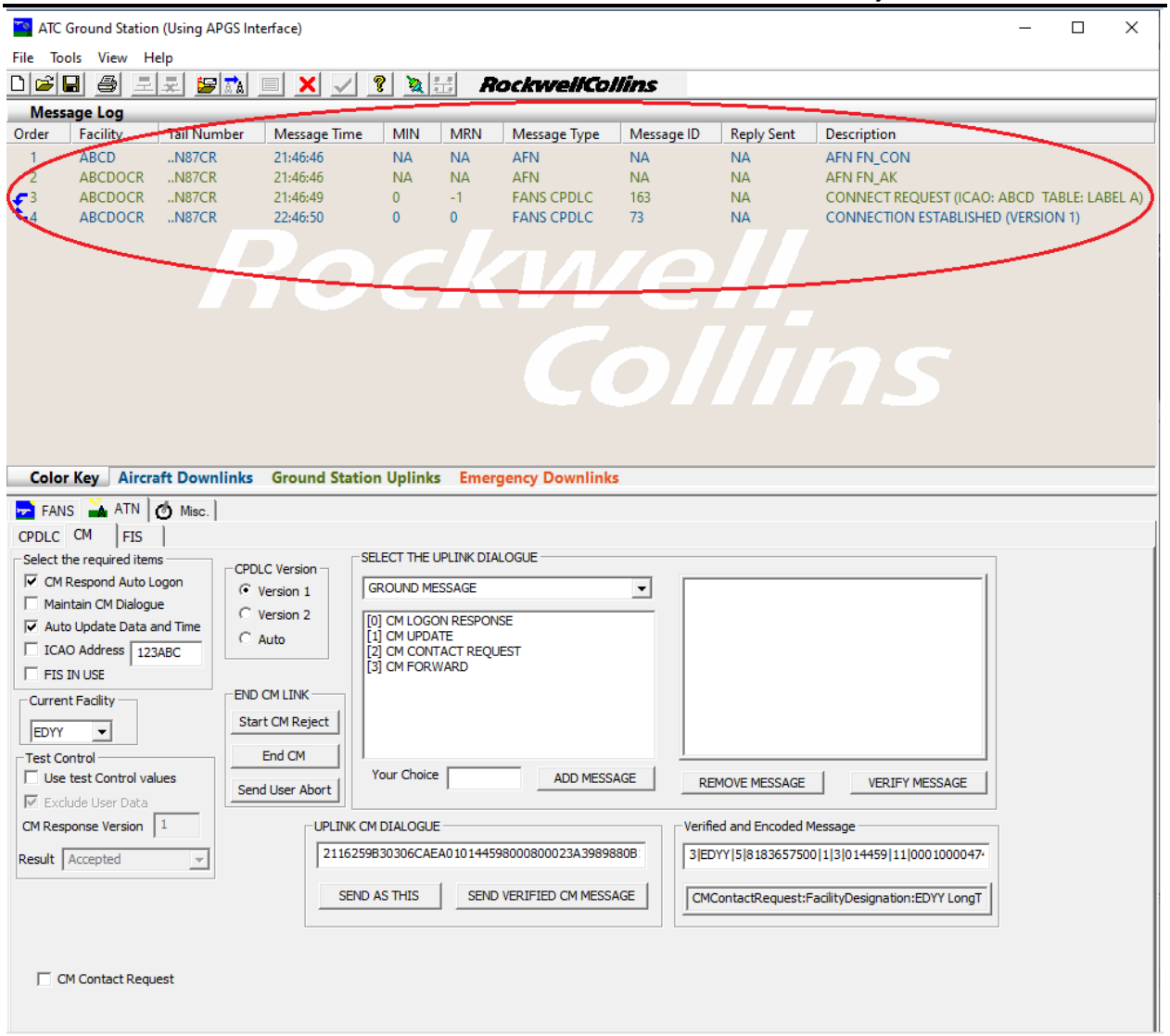


4. Click on CPDLC Icon and select Done.

6. Once the CPDLC page is opened, select the LOGON TO entry field and from L_MKP enter the value "ABCD" and press ENTER button on L_MKP.

7. In L_AFD window, click on SEND LOGON button, LOGON/STATUS PAGE on L_AFD is displayed with the status as below

8. Go to the ATC Ground Station application and you should see the below downlinks which confirms that you are able to perform a logon to the ground.



Once the logon is performed, Uplinks/downlinks can be send and tester can perform the required operations.

8.4 Target-based Test Station(Single DLCA)

The target based test station will be used to verify that Host results and target results are identical.

8.4.1 IPS Target

The Lauterbach dataload method is used for loading DLCA build onto target test stations. The Lauterbach dataload is an industry standard way to load software onto a target platform. It requires a preparation step prior to loading.

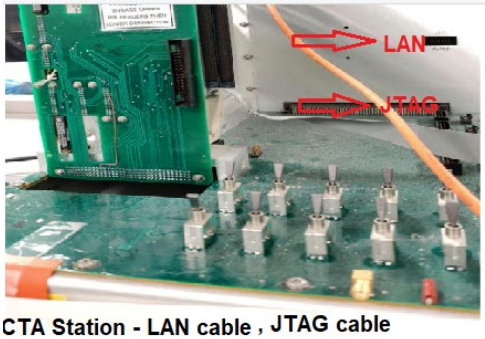
8.4.1.1 Physical connection details

To begin with, the following cables and equipment is required to set up the single DLCA target system. Serial cables – 3 , Ethernet cables – 2 , JTAG -1.

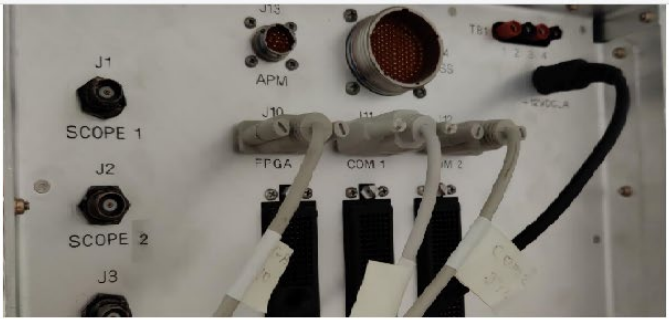
- Connect the serial cable coming from the FGPA port of the CTA station to the target PC
- Connect the Two serial cables - COM1 and COM 2 of the CTA station to the target PC
- Connect the ethernet cable coming from LAN A/LAN 1 port of CTA station to the ethernet hub/switch.

- Connect the ethernet cable from ethernet hub/switch to the target station.
- Connect the JTAG from the CTA station to the target PC

Refer the below images for connection details:

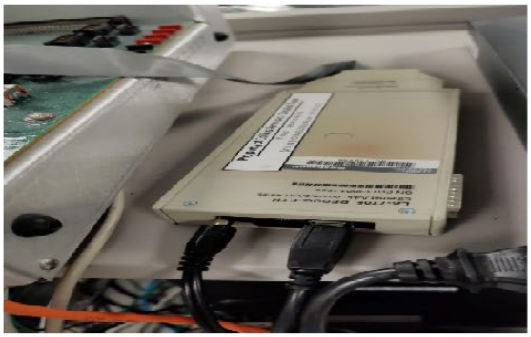


CTA Station - LAN cable , JTAG cable



CTA Station - FPGA, COM1, COM2 cables

Ethernet Switch/hub

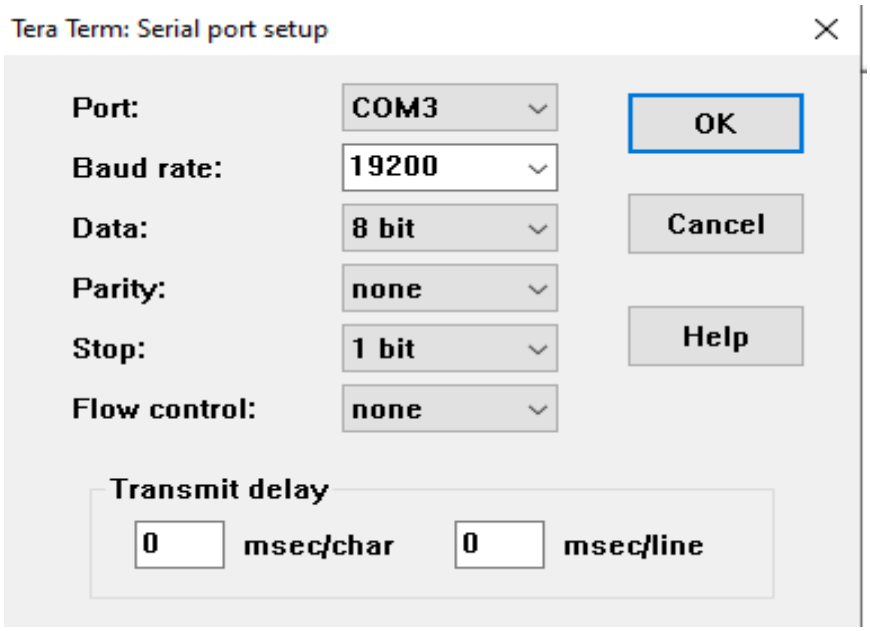


JTAG

IPS Target PC:

8.4.1.2 Loadset Preparation for Lauterbach dataload

1. Ensure that target station is installed/loaded/configured to support COM3, COM4, CPA-2079 Discretes , Trace32, FilterDrivers, TeraTerm before starting target execution. Refer <Verification-Branch>/tools/Target_Tools for the tools.
2. Open TeraTerm for COM3 and COM4, configured as shown below (depending on the test station configuration, COM3 and COM4 may be used instead).



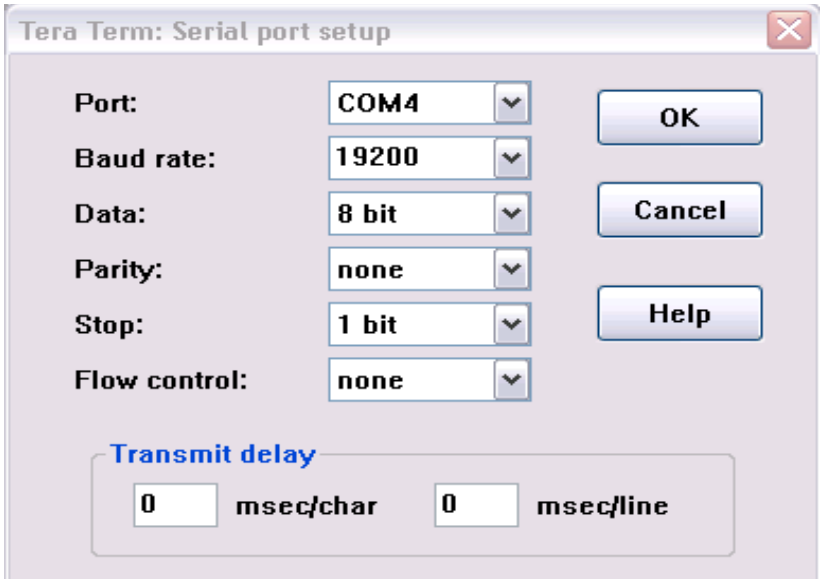


Figure 41: Tera Term:Serial port setup

3. Open two TeraTerm terminals, one for each of the VM's on the load. Typically PM and HM will output to COM3 and DLCA will output to COM4

   COM 3:

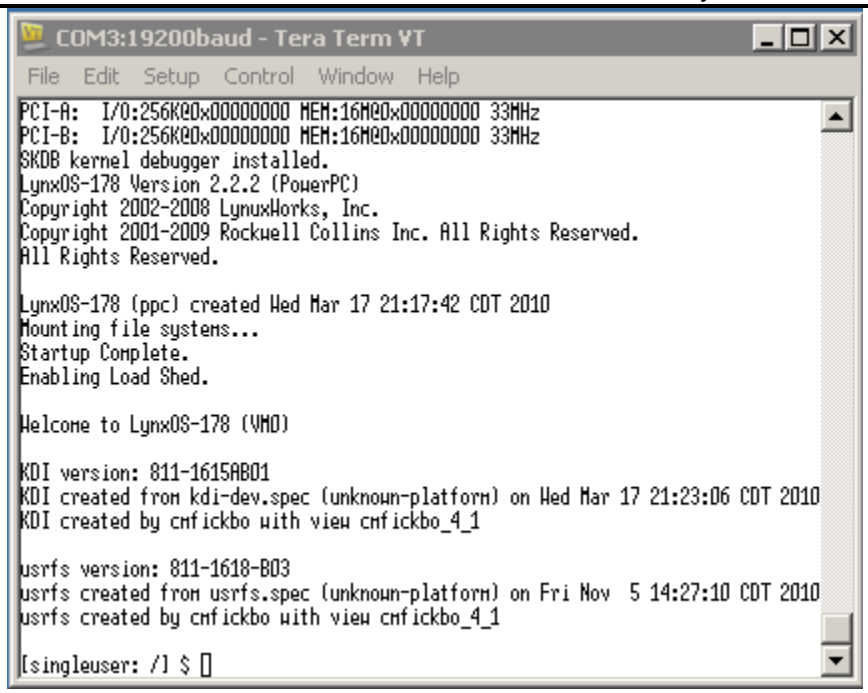


Figure 42: Tera Term – COM3 VT

COM 4:

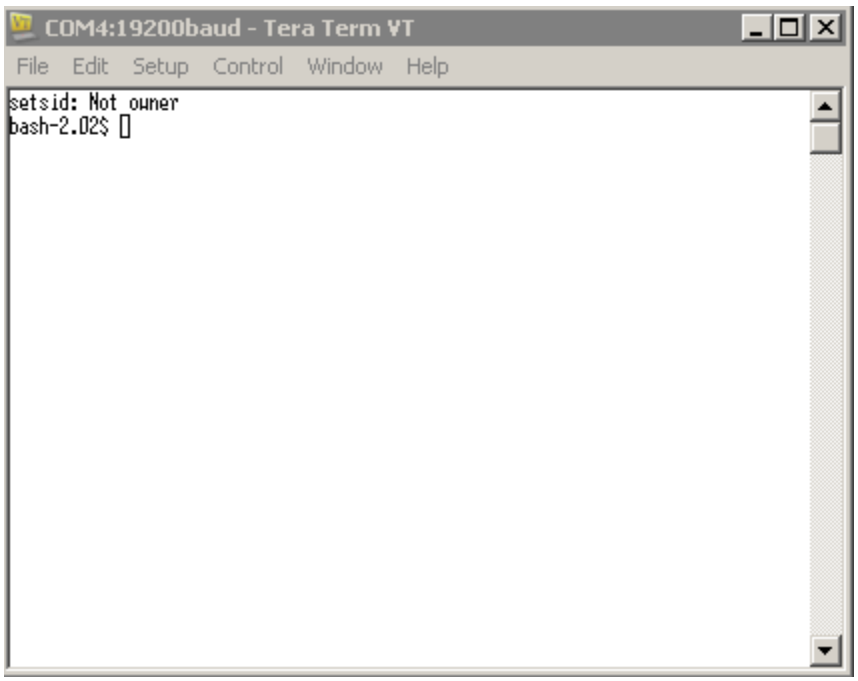


Figure 43: Tera Term – COM4 VT

4. Open <Verification-Branch>/tools/Target_Tools/CPA_Discretes_CTA2079/bin/cpa_disc_cta2079.exe and configure as shown below.

   CPA Discretes

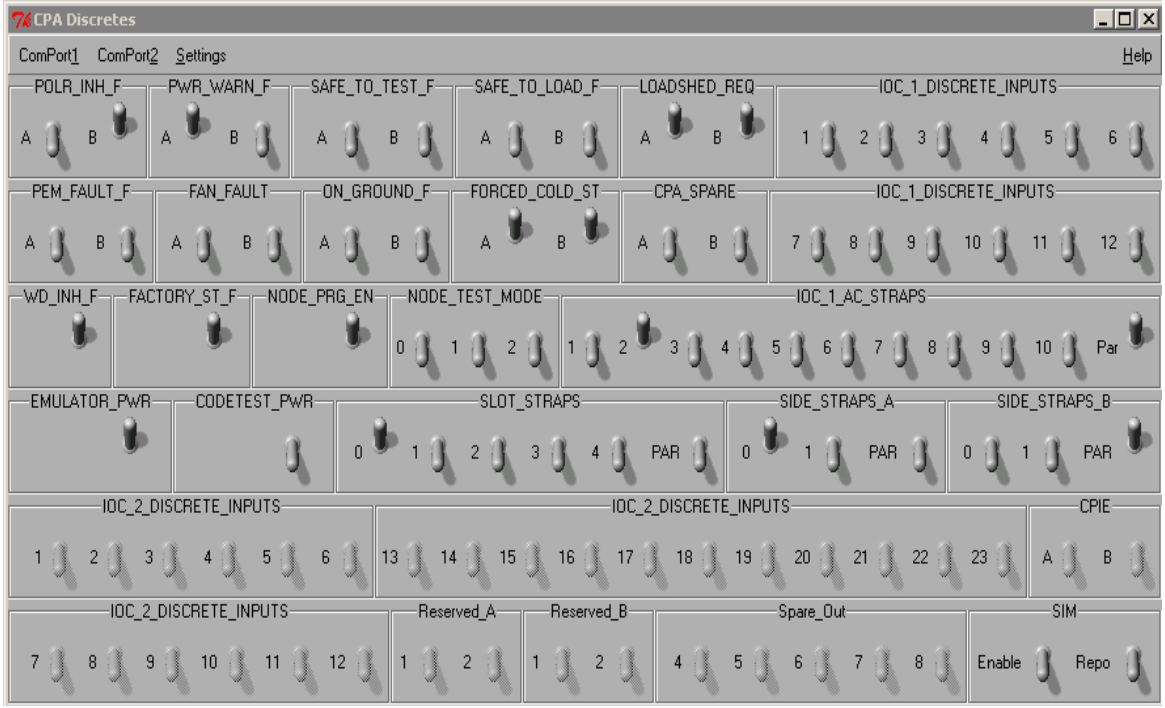
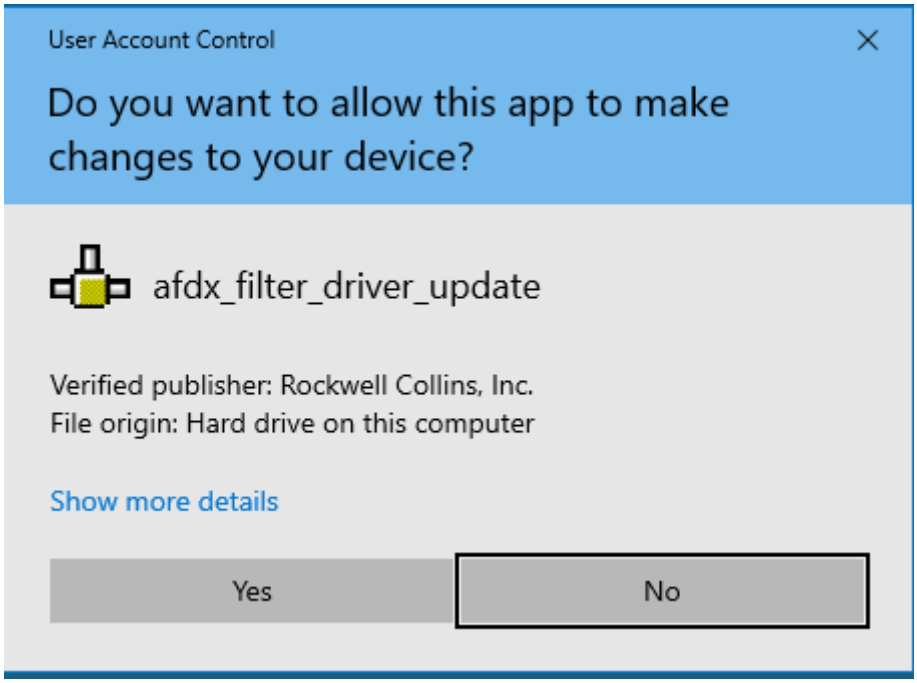


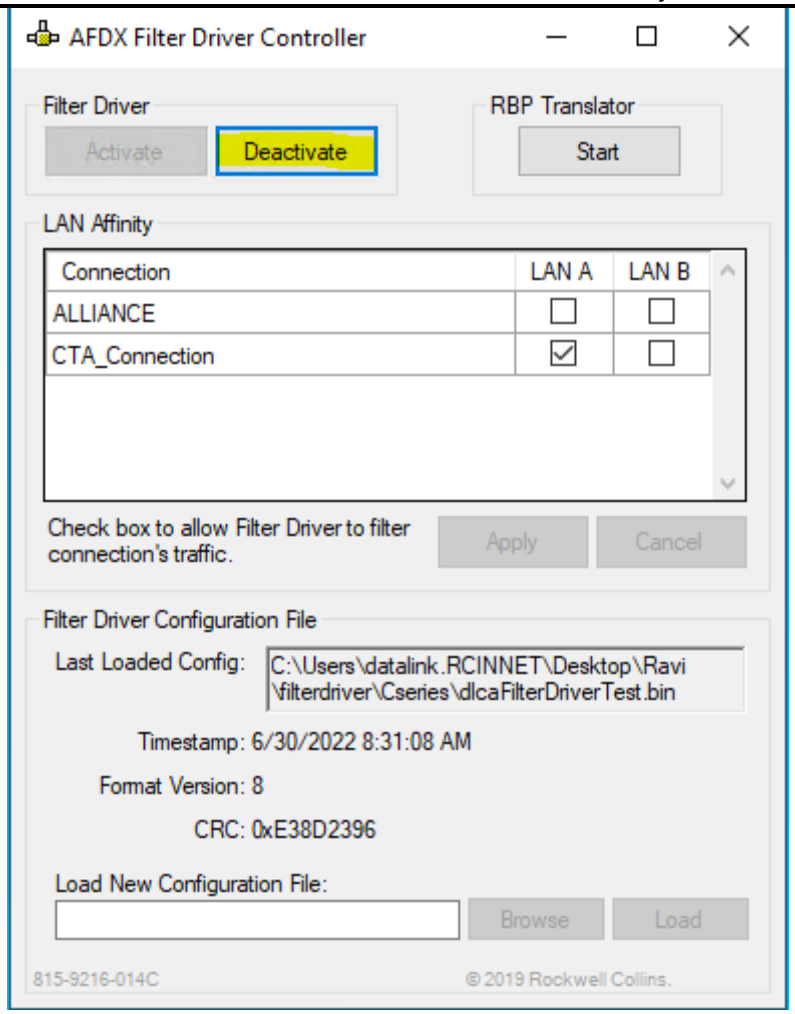
Figure 44: CPA Discretes

5. FilterDriver Settings:

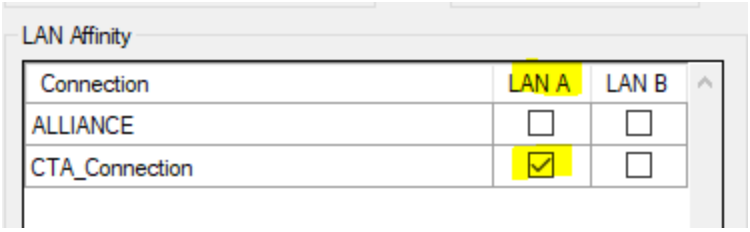
5.1 Launch the AFDX Filter Driver Controller short cut from desktop and Select 'Yes'



5.2 Select Deactivate button in AFDX Filter Driver Controller window .



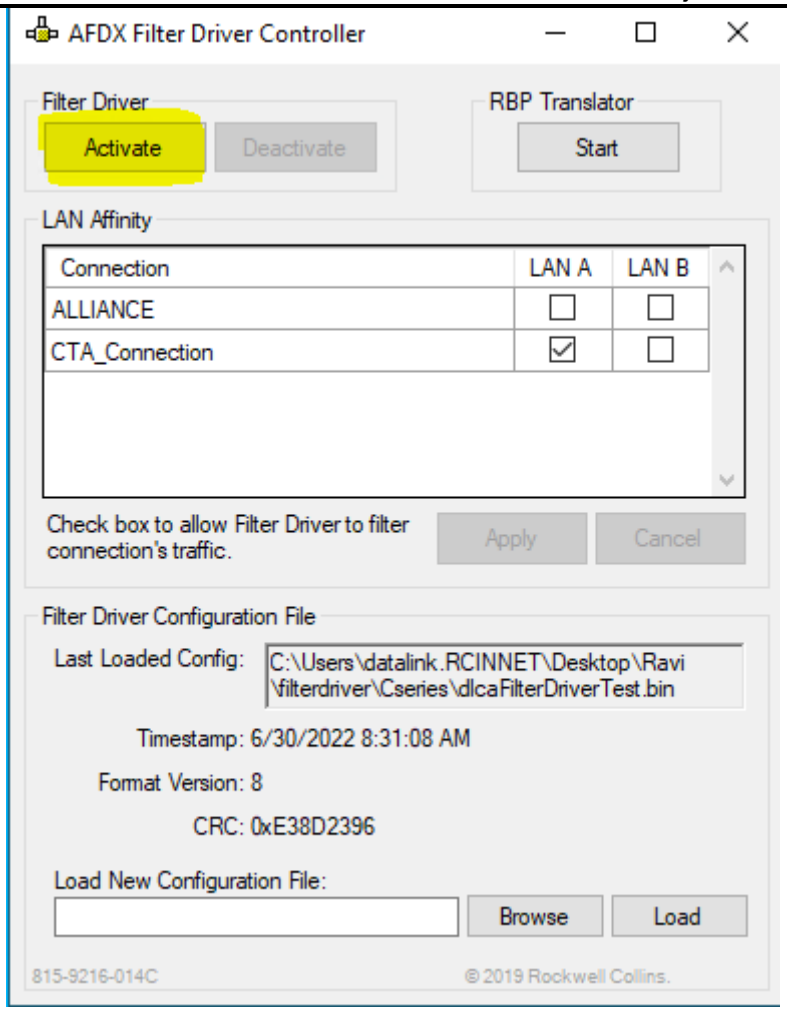
5.3 Select the correct LAN connection and select LAN A

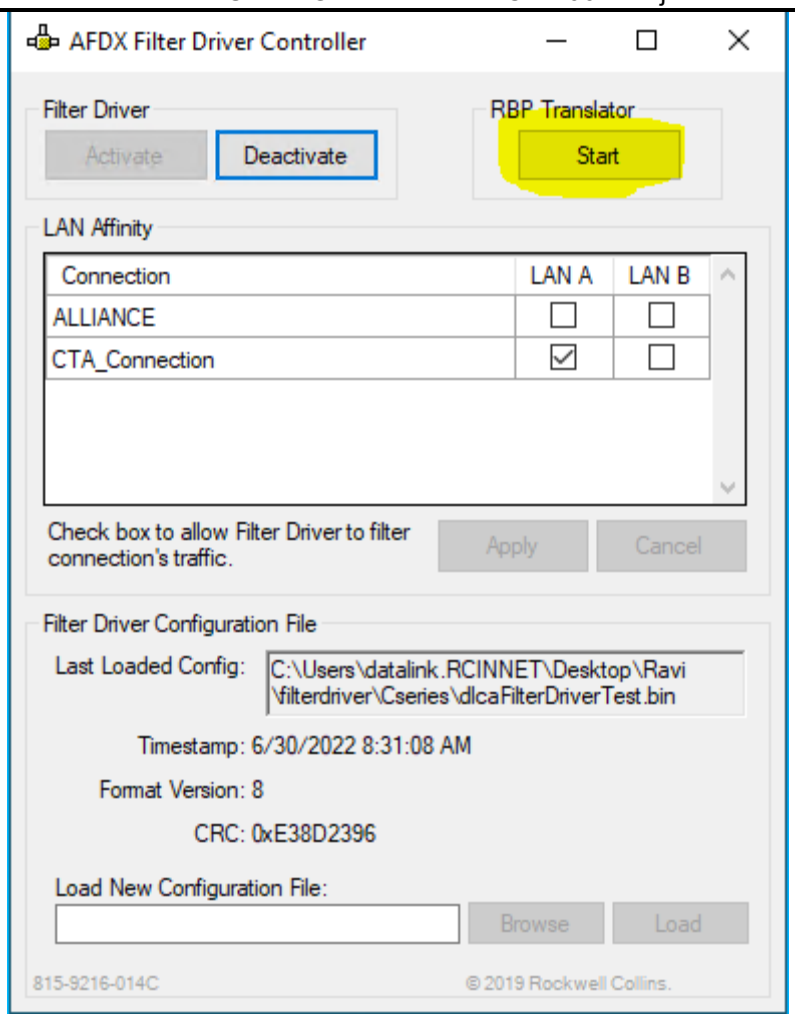


Browse the DLCAfilterdriver.bin file from below SVN path and load the file 'dlcaFilterDriverTest.bin'

<Verification-Branch>/tools/Target_Tools/FilterDrivers/filterdriver/{ProgramName}
for example {ProgramName} is Cseries.

5.4 select Activate button to install the driver. Select the RBP Translator 'Start' button and Wait util Server starts





5.5 After seeing below message, close command window and AFDX Filter Driver Controller windows.

8.4.1.3 TTL files for Target

Ttl files are used to automate Tera Term. All ttl files are available in <Verification-Branch>/vista_sim/scripts. For IPS environment, other applicable simulations are <Verification-Branch>/vista_sim_GS, <Verification-Branch>/vista_sim_IPSDual, <Verification-Branch>/vista_sim_MRJ

A_PM_CREATE.ttl

This is a teraterm macro file to send start process commands using teraterm communication port to LEFT PM.

A_PM_KILL.ttl

This is a teraterm macro file to send kill process commands using teraterm communication port to LEFT PM.

L_DLCA_CREATE.ttl

This is a teraterm macro file to send start process command using teraterm communication port to LEFT DLCA.

L_DLCA_KILL.ttl

This is a teraterm macro file to send kill process command using teraterm communication port to LEFT DLCA.

L_DLCA_NVM_PURGE.ttl

This is a teraterm macro file to send command nvm_purge using teraterm communication port to LEFT DLCA.

LOG_CREATE.ttl

This is a teraterm macro file to capture the statements printed using teraterm communication port to LEFT DLCA.

8.4.1.4 Lauterbach load

1 To perform a Lauterbach load, a target build will need to have been successfully completed as per the guidance in the section 'Custom Builds Instructions', sub section IPS Target Build .
2 Create a folder in the target pc, and place localfs_<IPS_program name> generated as per section 6.1 IPS Target Build, for IPS Target Build. Similarly refer section 6.2 IPS Custom Builds for IPS Custom Builds , section 6.3 Timing Build, for Timing Build. For example create folder as U:\CSeries_dlca\ and place "localfs_cseries" build file for IPS Target Build.
Note: For dry-runs and RFS, get the latest builds from
For normal build: <Dev-Branch>\Build\releases,
For custom build: <Verification-Branch>/Reference/Custom_Build_Support/Builds/x.x.x
For Timing build: <Verification-Branch>/test_procedures/TIMING/Mediasets/x.x.x
where x.x.x is latest build version folder
3 Open cpa_disc_cta2079.exe, as shown and ensure that the ComPort1 is select as COM1, and ComPort2 is disabled.

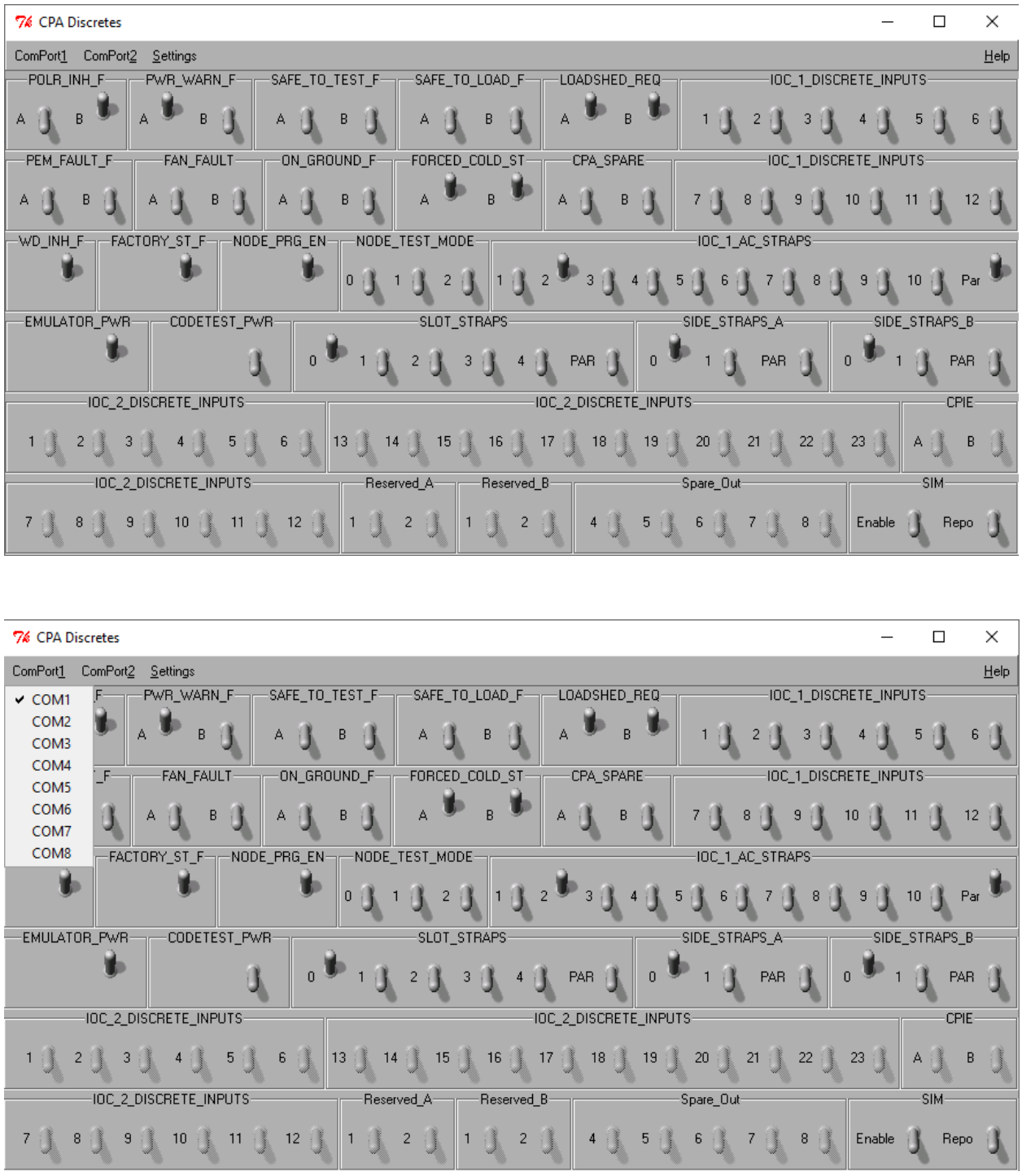
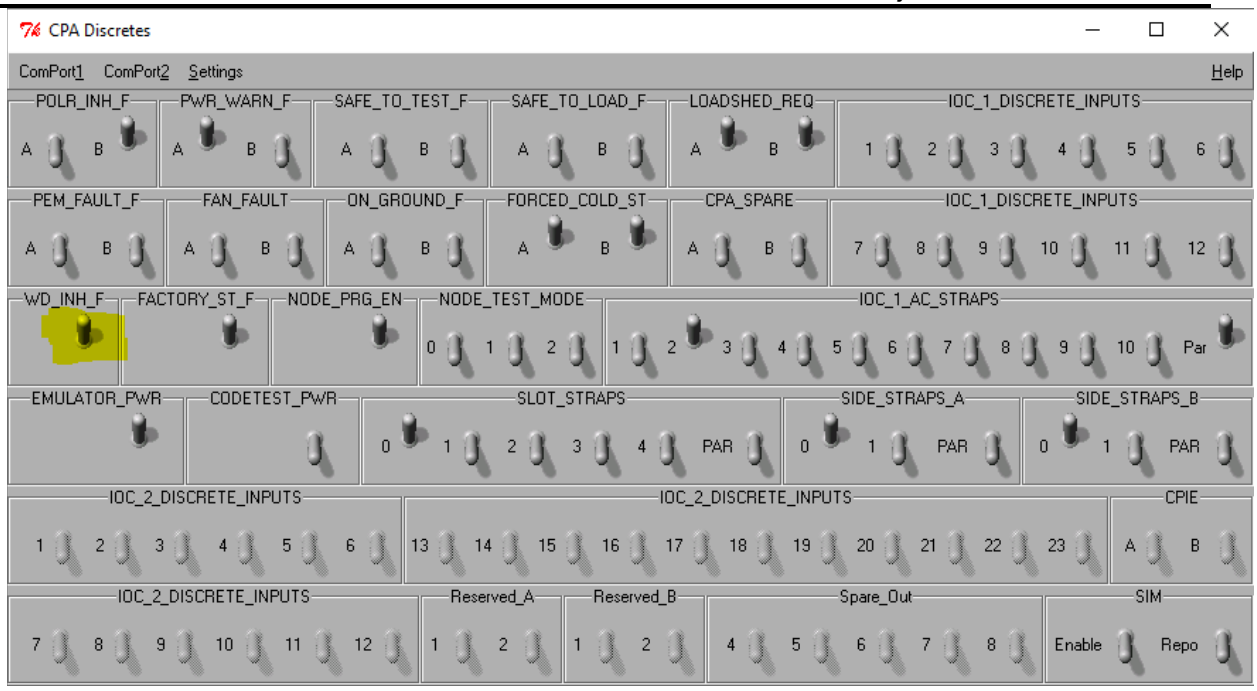
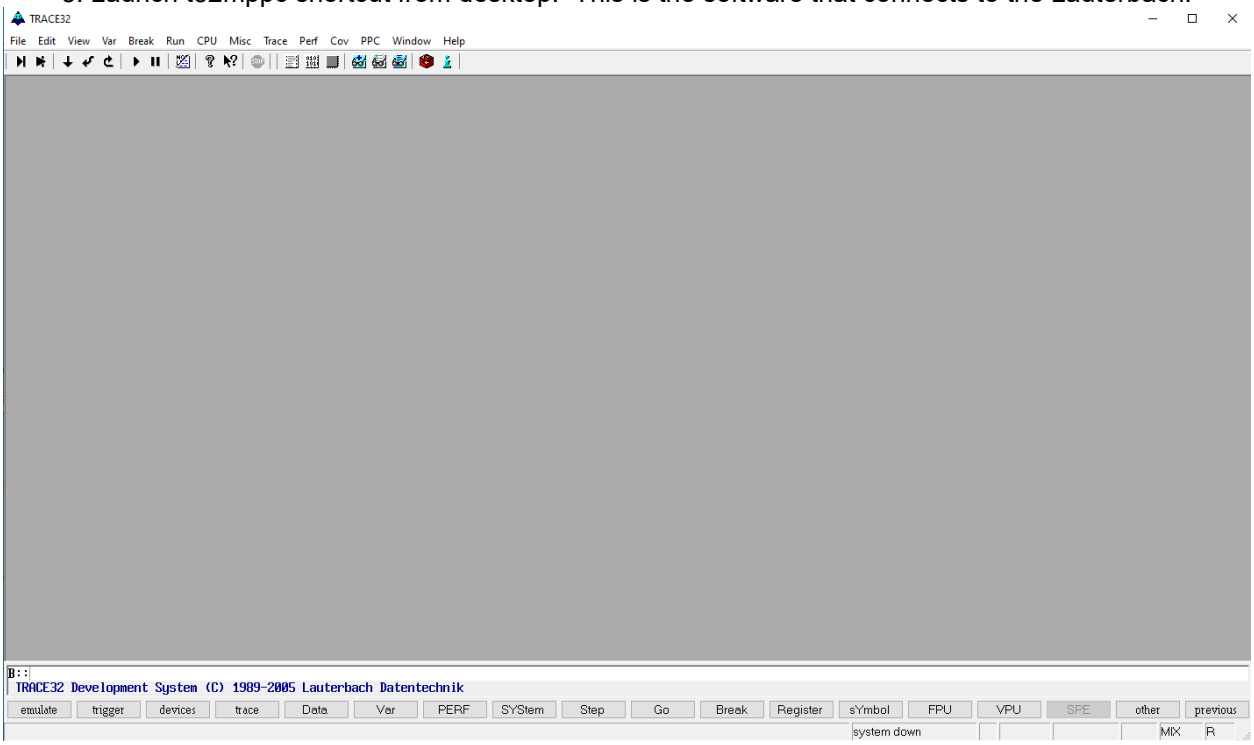


Figure 45: CPA Discretes

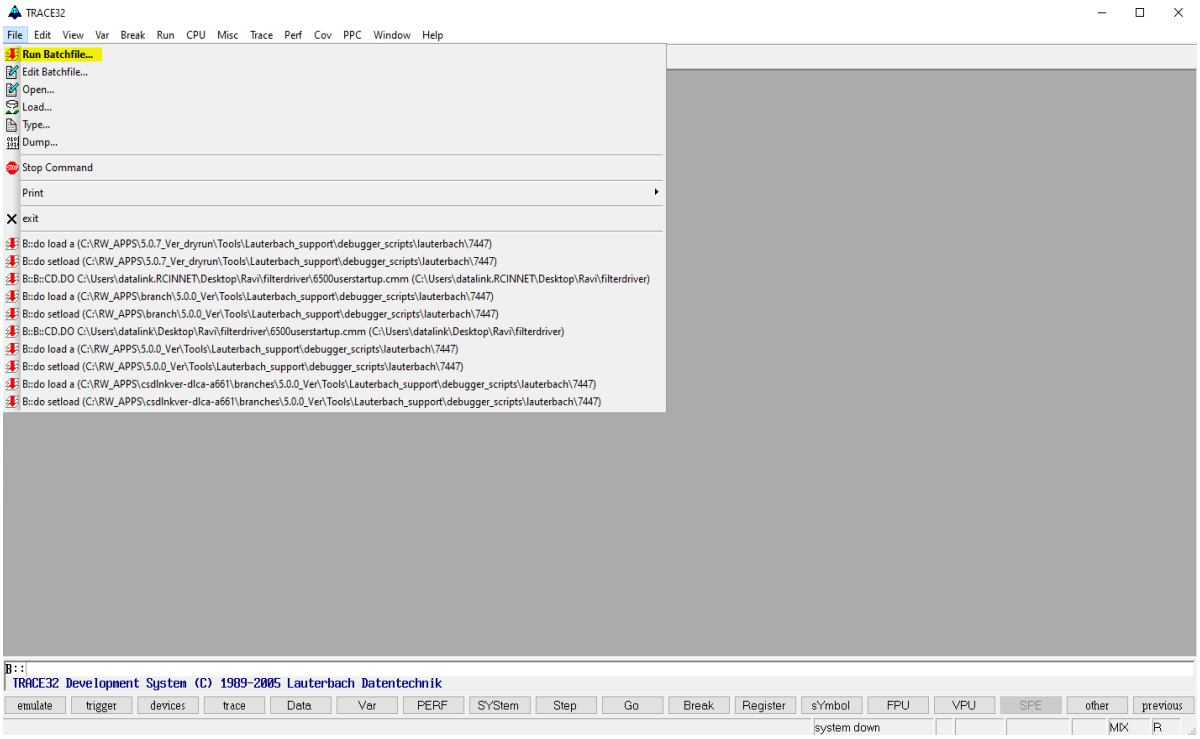
4 Ensure that Turn watchdog (WD_INH_F) is on.

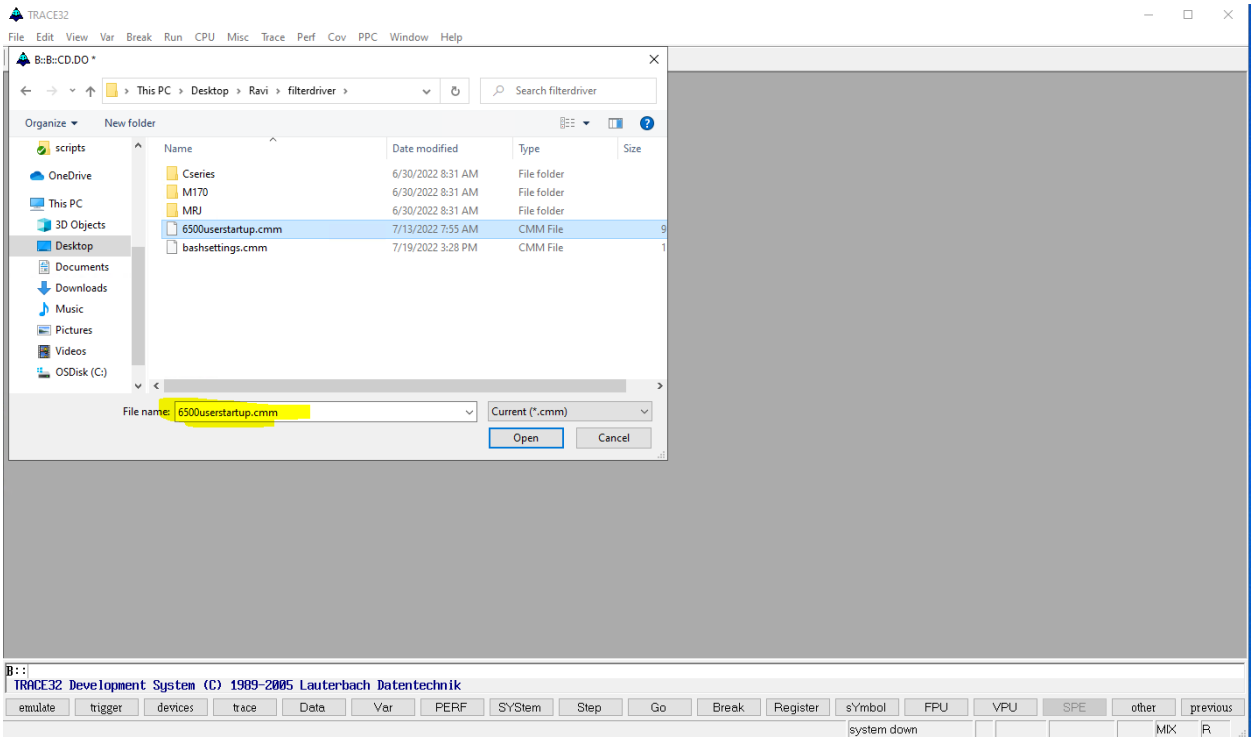


5. Launch t32mppc shortcut from desktop. This is the software that connects to the Lauterbach.

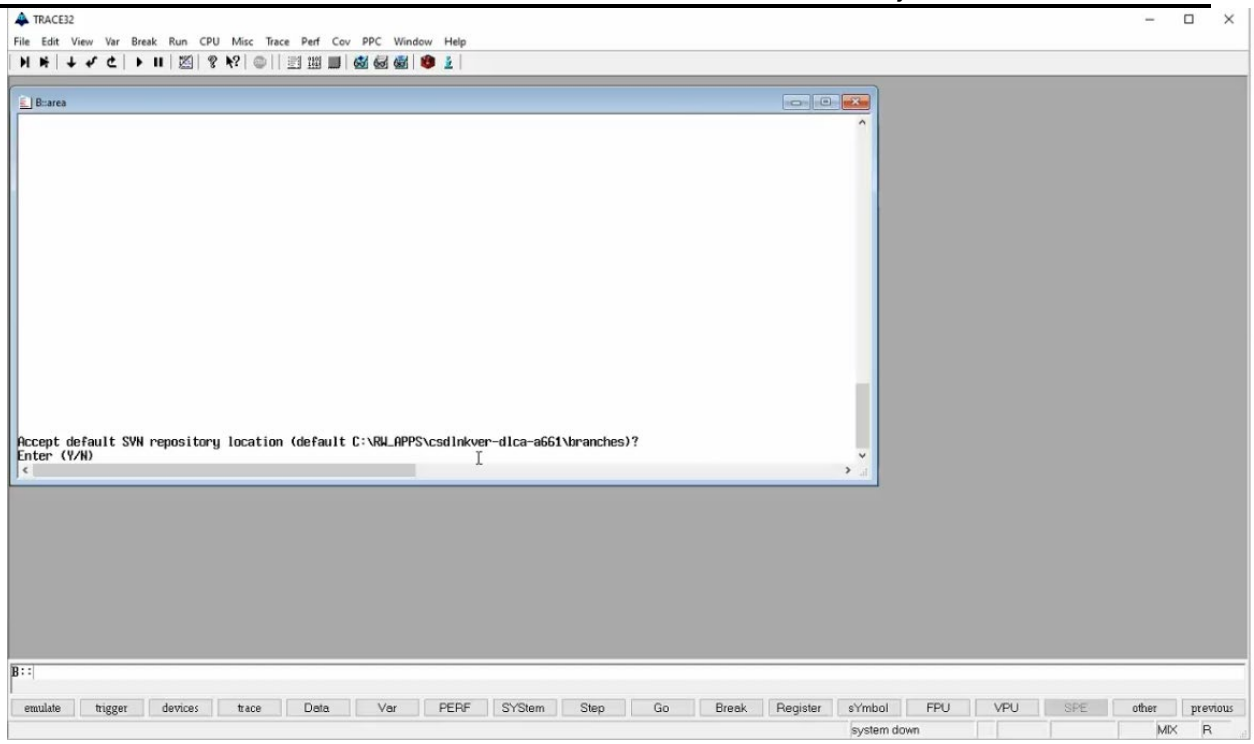


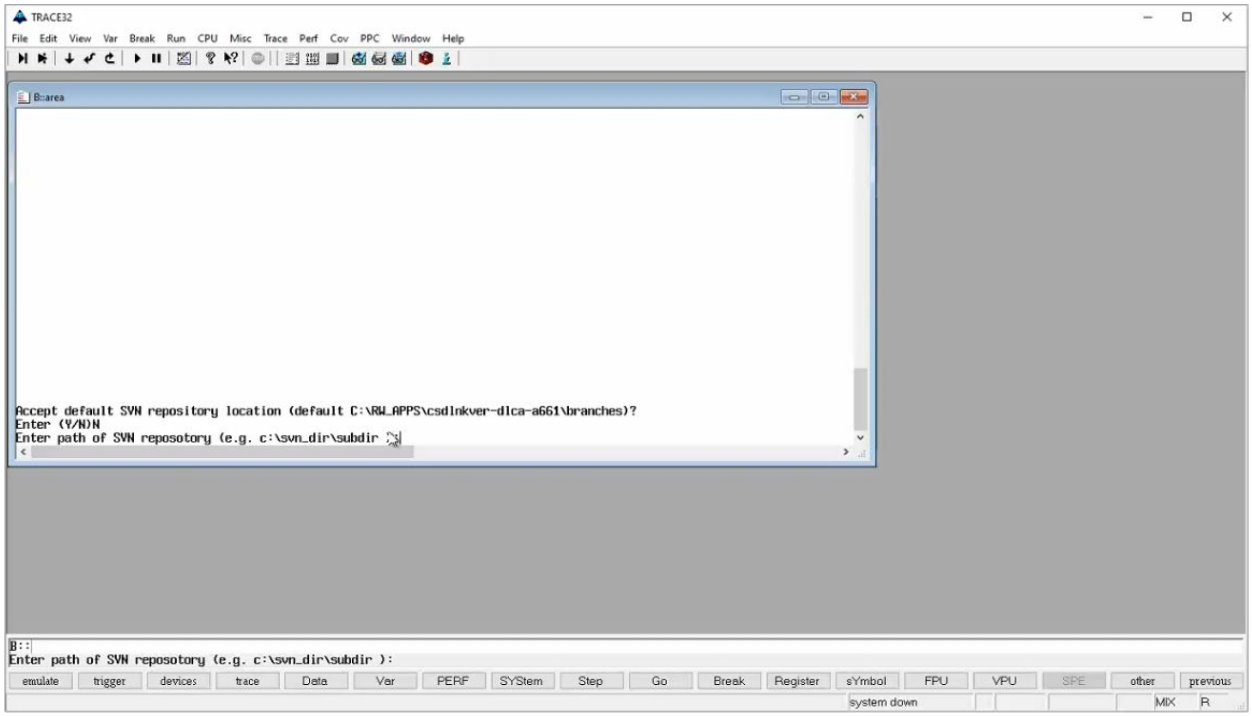
5 Run the "<Verification-Branch>/tools/Lauterbach_support/6500userstartup.cmm" file from File->Run Batchfile

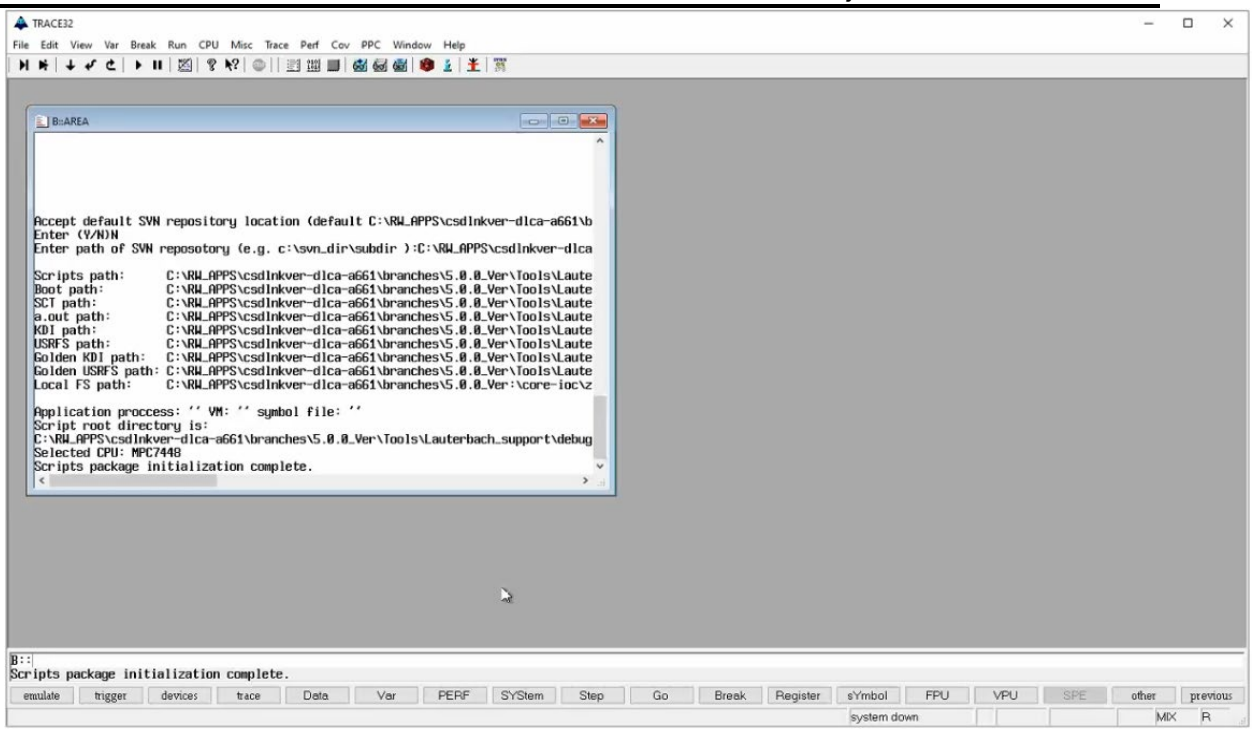




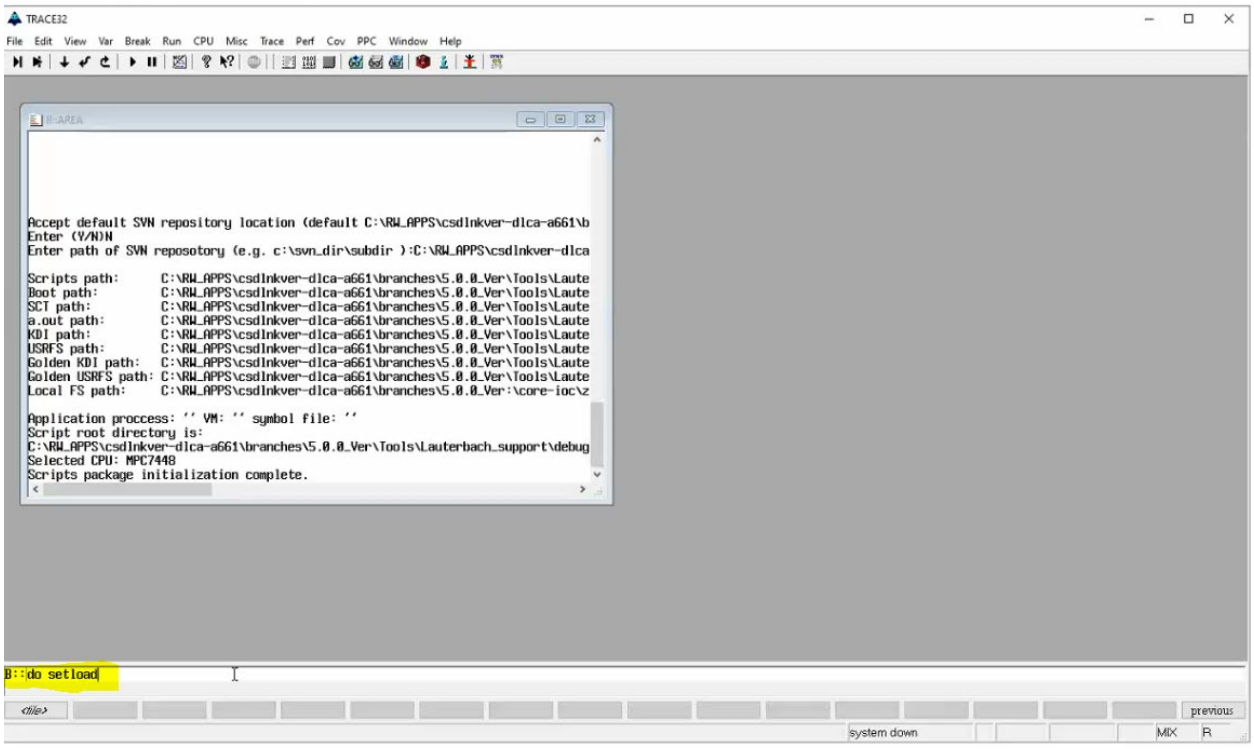
6 Enter 'Y' if the default SVN repository location shown is correct or Enter 'N' and Enter path of local drive which is checkedout from SVN repository https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/.





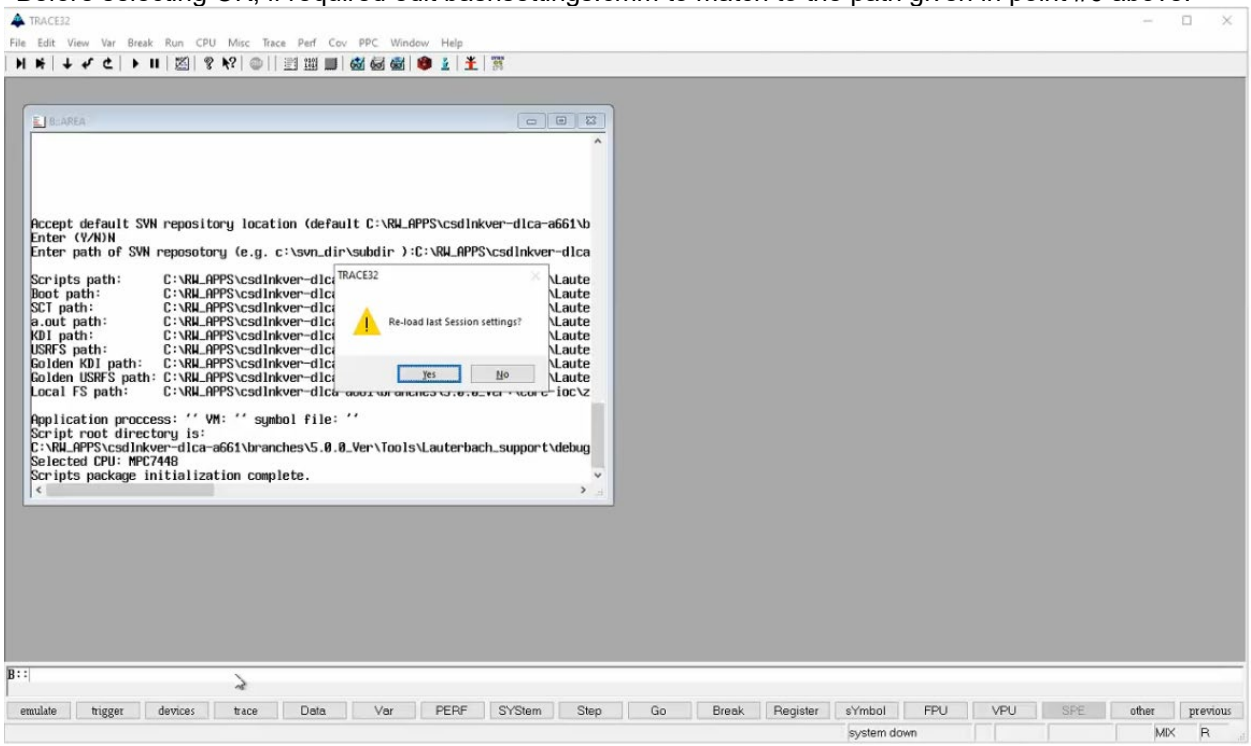


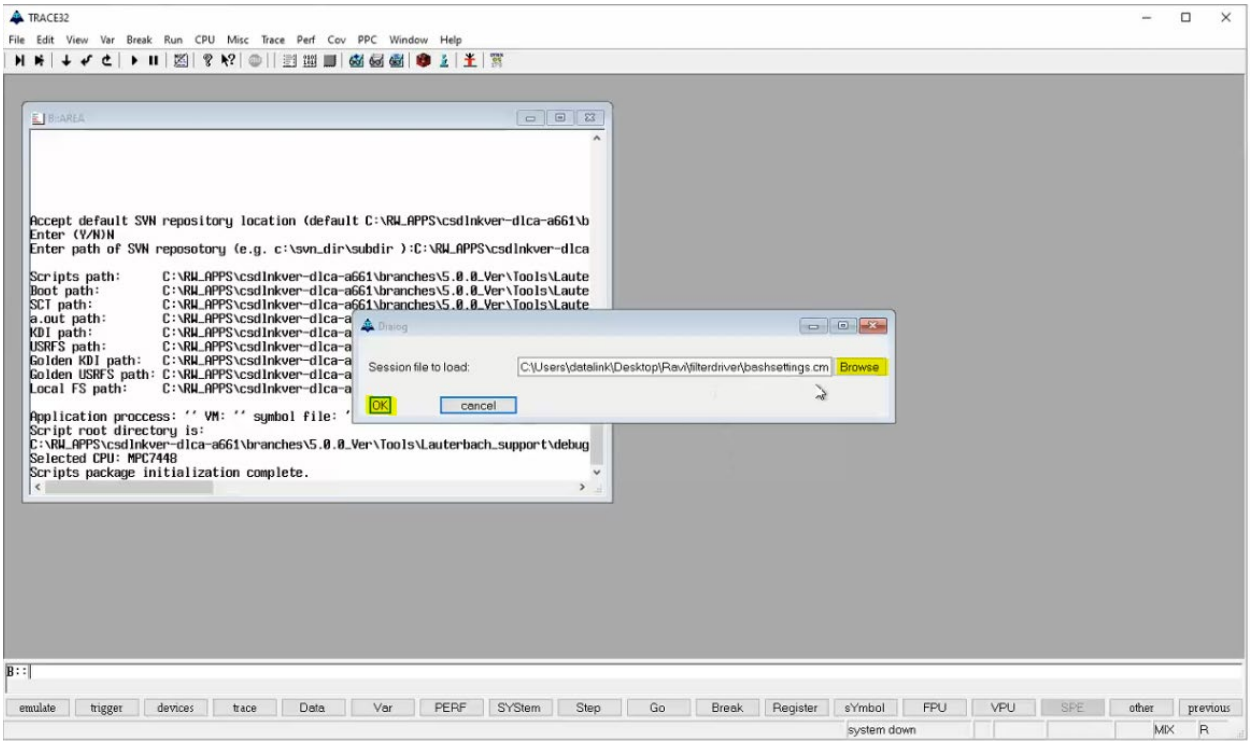
7 After seeing message, 'Scripts package initialization complete', Execute 'do setload' from the Trace32 command prompt.



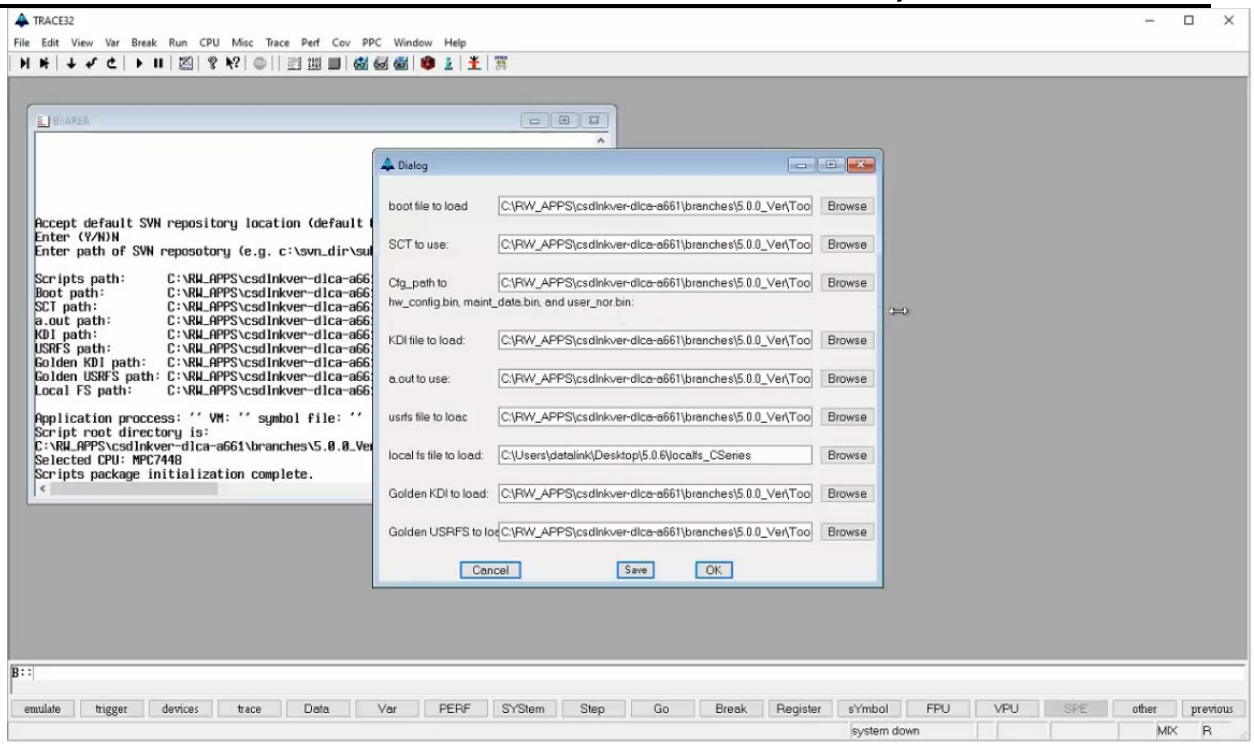
8 Choose 'Yes' to the Reload last Session settings popup, and browse to the setttings file bashsettings.cmm[checkedout from <Verification-Branch> /tools/Target_Tools/FilterDrivers/filterdriver/bashsettings.cmm], and select OK.

Before selecting OK, if required edit bashsettings.cmm to match to the path given in point #6 above.

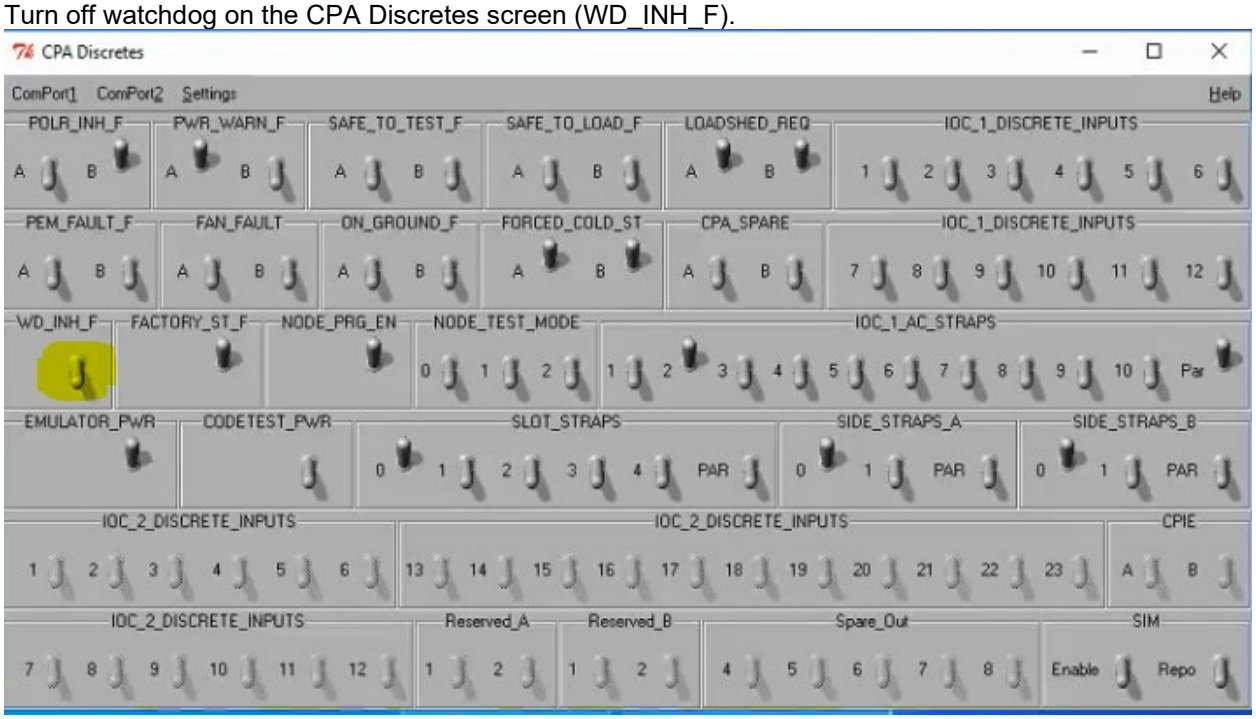




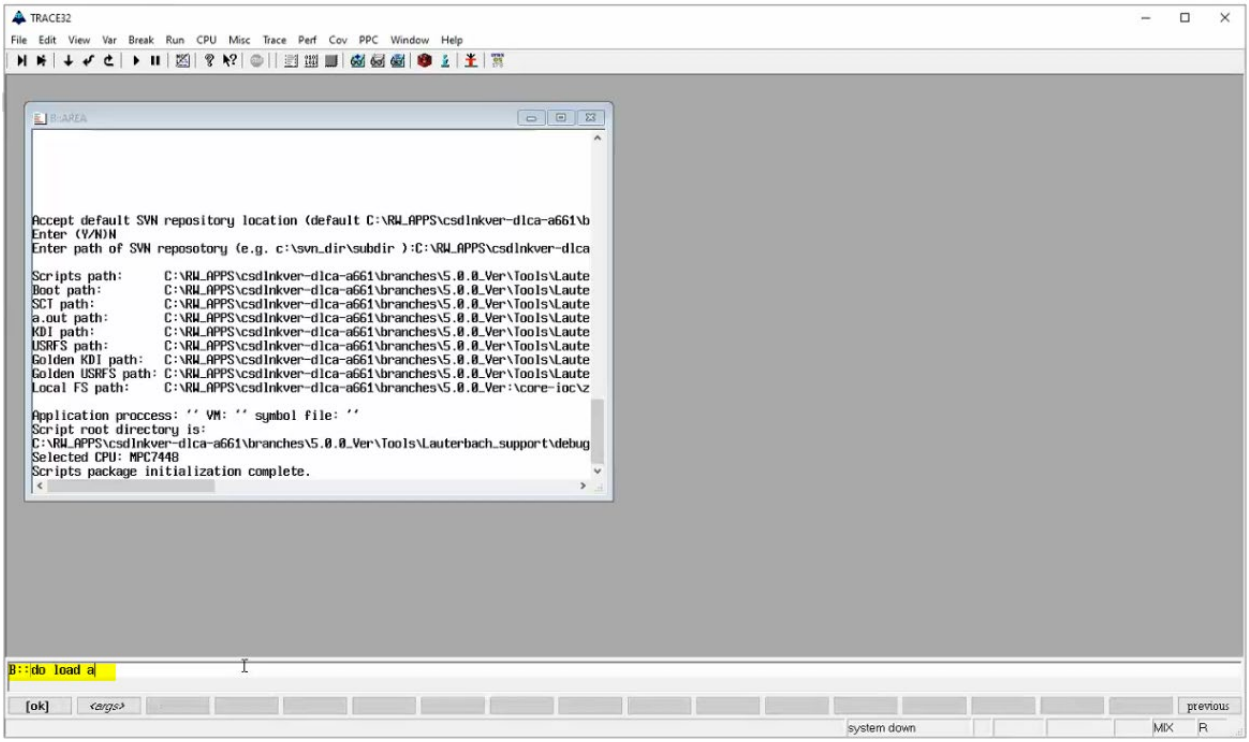
10. Confirm the proper files are selected. There are a couple of things to note on this screen. Ensure that kdi, a.out and usrfs are pointing to dev versions, as prod versions will not work with BASH prompt loads. Check 'local FS' is pointing to the correct location for the load you are doing. Currently there are unique locations for different project's which loads. Select OK.



11. Turn off watchdog on the CPA Discretes screen (WD_INH_F).



12. Execute 'do load a' from the Trace32 command prompt.



13. When the screen looks like the following the, CCM is successfully loaded.
    Once the Load is complete close Trace32.

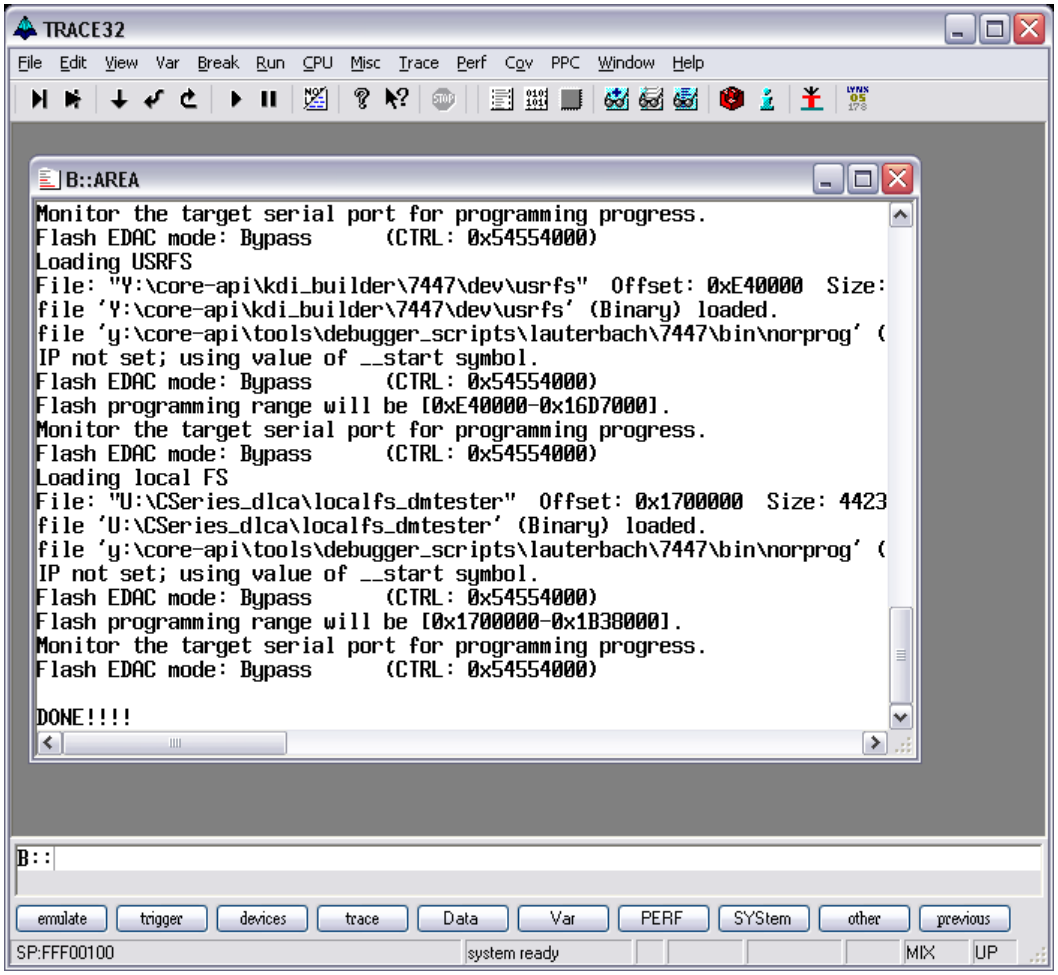
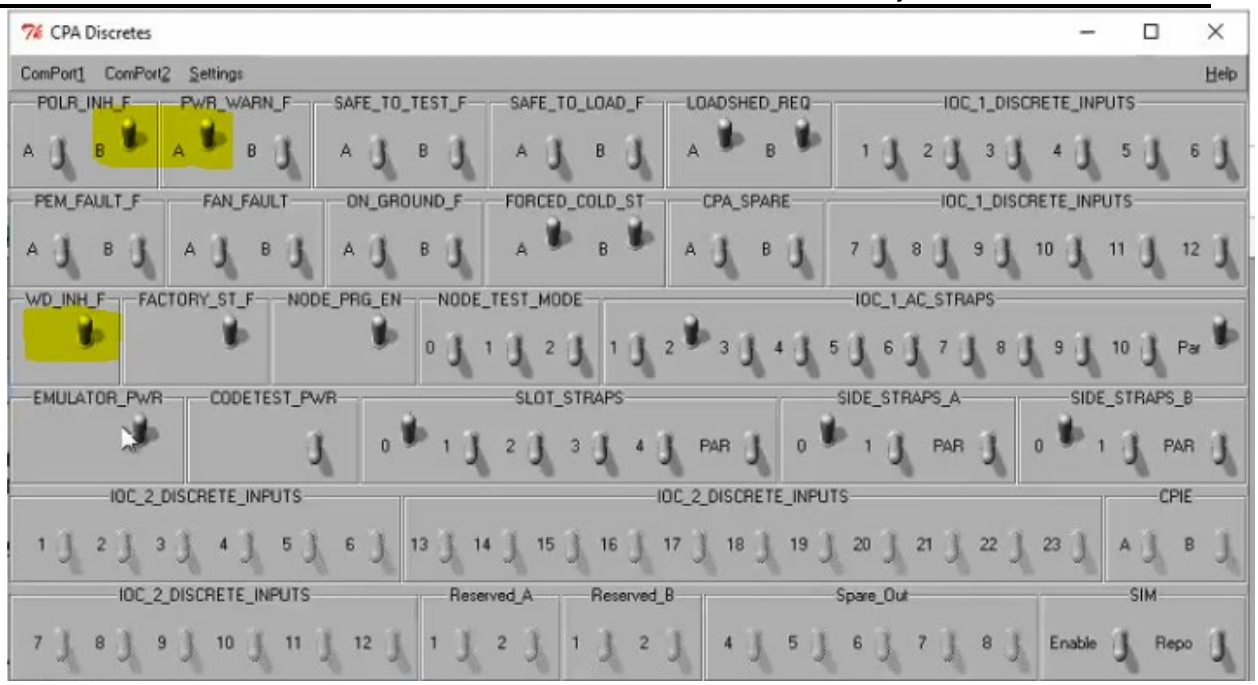


Figure 46: TRACE32 after CCM is successfully loaded

14. Open CPA discretes, Turn watchdog (WD_INH_F) back on. Power off the discretes PWR_WARN_F switch A and POLR_INH_F switch B.



15. Start target. Follow the instruction as per the below section "Starting the Target"
16. After initial loading the DLCA will get reset as license and strapping data does not match the NVM, because NVM will be cleared after factory reset [As per DLCA requirements]. Therefore DLCA has to be restarted. After restart, license and strapping matches the NVM thus DLCA will be up and running.

8.4.1.4.1 Starting Target

1. Open COM3 and COM4 teraterm windows.
2. Power On the discretes PWR_WARN_F switch A and POLR_INH_F switch B
3. Enter below commands without quotes, by right clicking on bash command prompt in teraterm console COM3, at '[singleuser: /] $'
   "sw_val_complete"
   "cinit cnc"
4. Enter 'pm' in the TeraTerm console to start Protocol manager.
   In COM4 Tera Term window,
5. Enter 'dlca' in the other console to start DLCA. The startup process will initialize all of the data flows and return 'Init done' when successfully completed.Make sure that the partnumbers are correct or not.After this, Close all tera term and cpa discrete windows.
   Note: Bash prompt will come for 3 times in teraterm console, if the commands are given properly With in these attempts, then steps 1 to 5 need to be followed untill successfully completed.
6. Start the VISTA environment from the local repository path, which has been checkedout from https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/vista_sim/System_target.bat directory of SVN to start all of the hosted applications that connect to DLCA and PM.
7. Check for DLCA and PM Health Status, to ensure DLCA is working. If it is off , Terminate the Vista environment and re-run A_PM_create.ttl and L_DLCA_create.ttl files.
   [Refer to section 8.4.1.2 TTL files for Target, for svn path of the files]

8.4.2 EDS Target

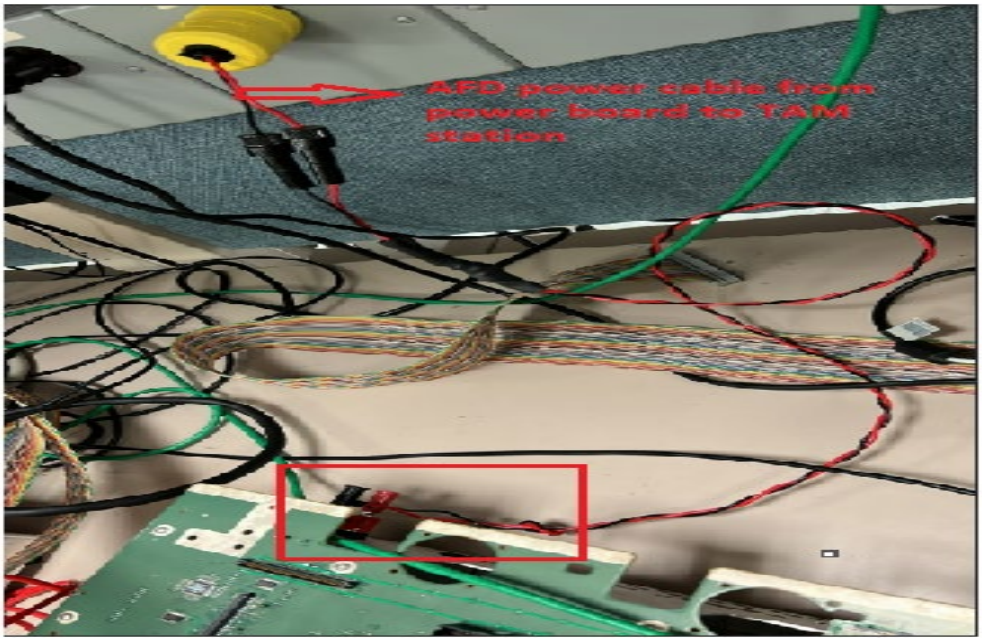
The 615A dataload method is used for loading DLCA build onto target test stations. The 615A dataload is an industry standard way to load software onto a target platform, utilizing the ARINC615 specification. It requires a preparation step prior to loading.

8.4.2.1 Physical Connection Details

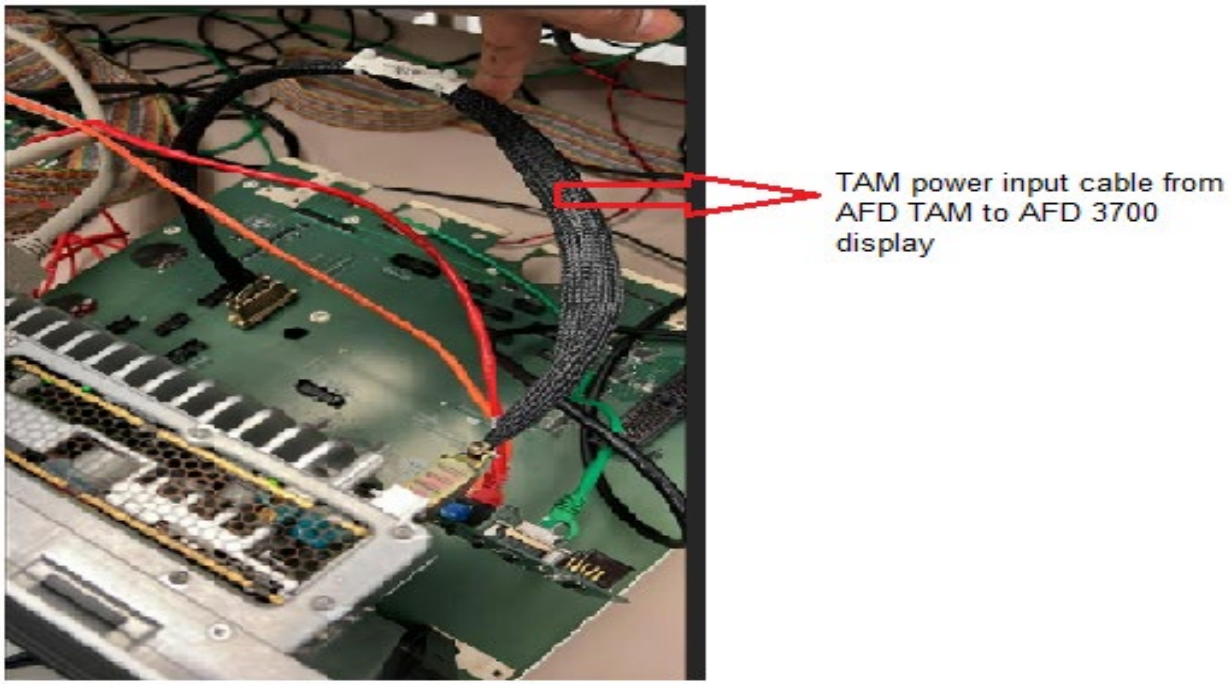
To begin with, the following cables and equipment is required to set up the dual target system.

8.4.2.1.1 Single AFD TAM Station:

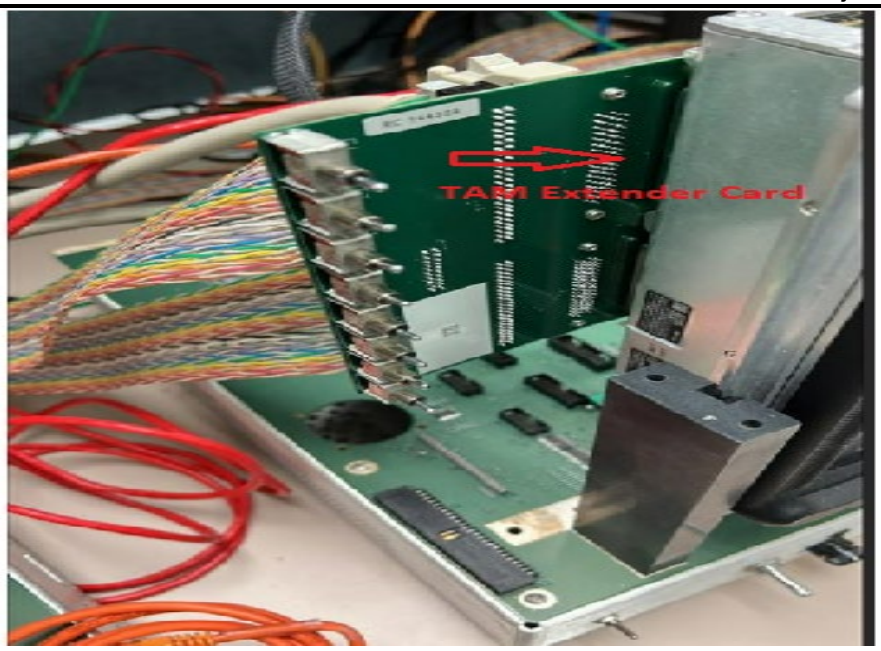
- One AFD power cable coming from power board and going to the AFD TAM station



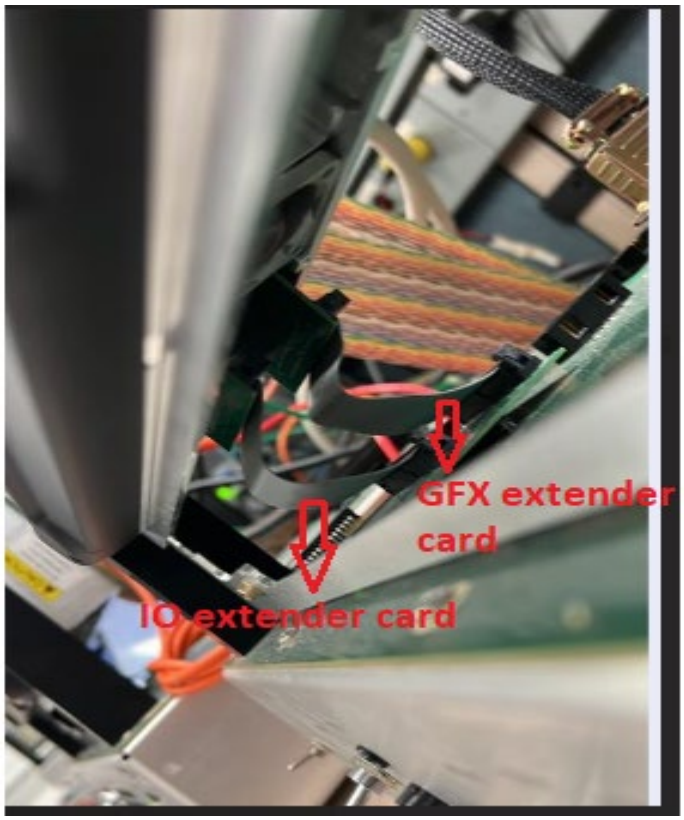
- One TAM power input cables coming from AFD TAM station and going to the AFD 3700 display



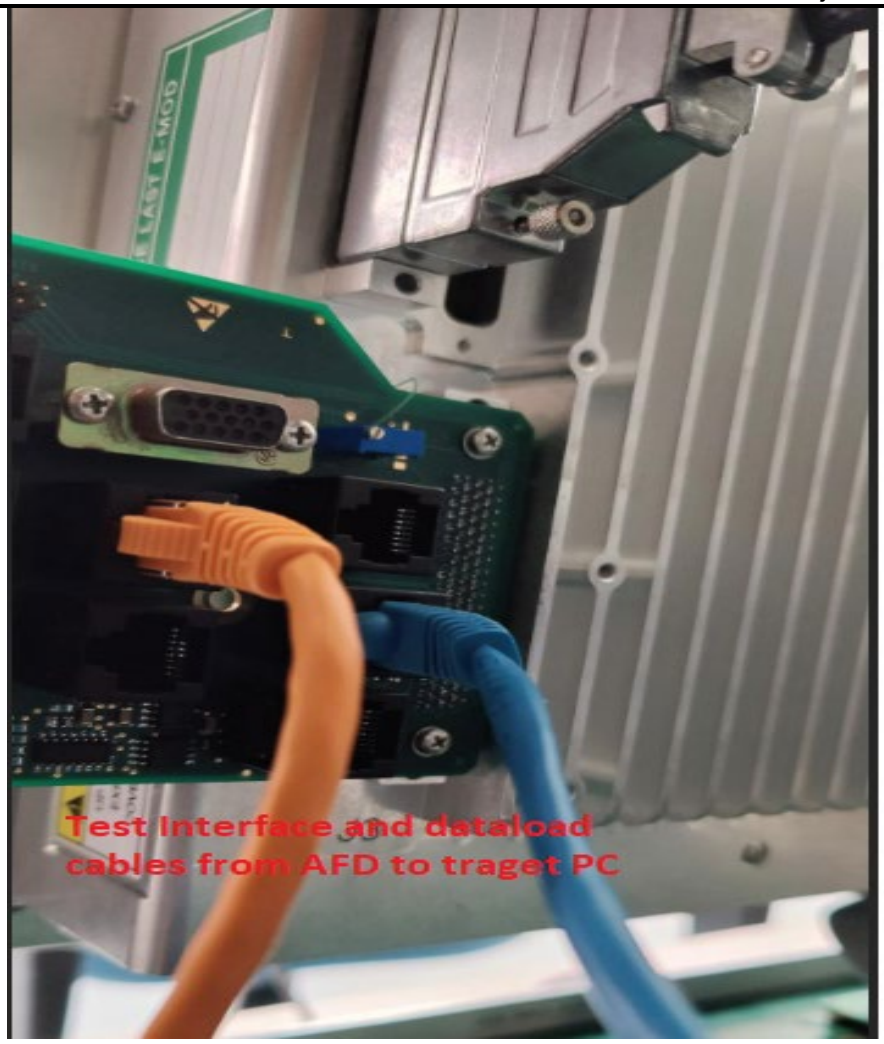
- One TAM extender Card coming from AFD TAM and going to the AFD 3700 display



- One IO extender Card coming from AFD TAM and going to the AFD 3700 display
- One GFX extender Card coming from AFD TAM and going to the AFD 3700 display



- One Dataload cable and one TestInterface cable coming from AFD J3 Wraparound board and going to Target PC.



- Assign the IP address and subnet mask value as per below:

  Dataload (COM5):

  IP address: 10.129.25.0

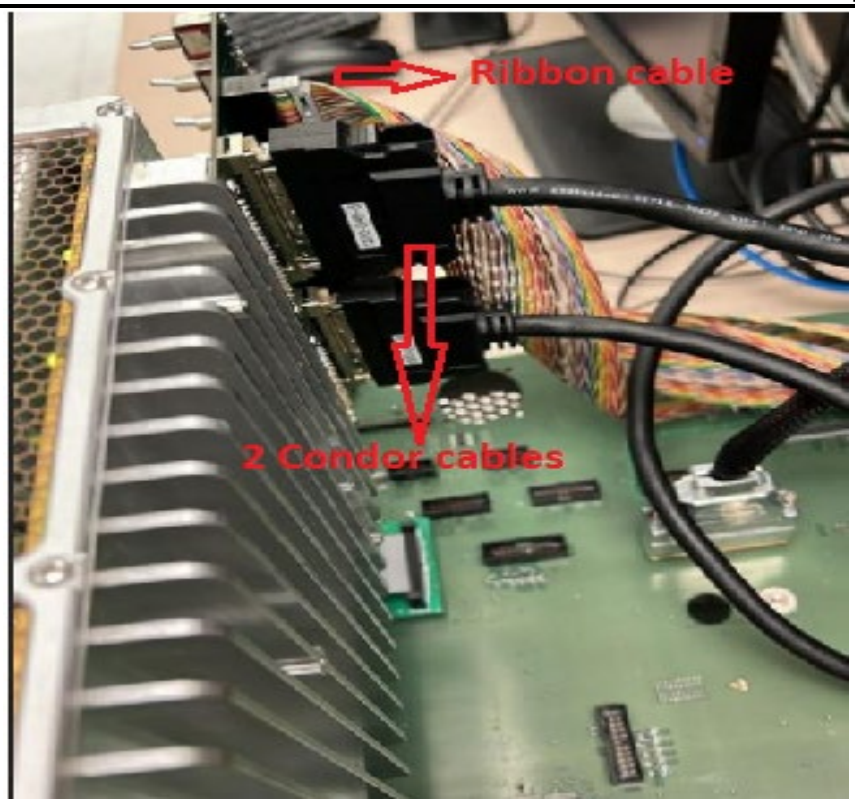
  Subnet mask: 255.224.0.0

  TestInterface(COM7):

  IP address: 10.145.26.1

  Subnet mask: 255.255.255.0

- Two Condor cables coming from the AFD TAM Discrete card and going to the serial ports on the PC (two per PC)
- One Ribbon cable coming from the AFD TAM Discrete card and going to the PC



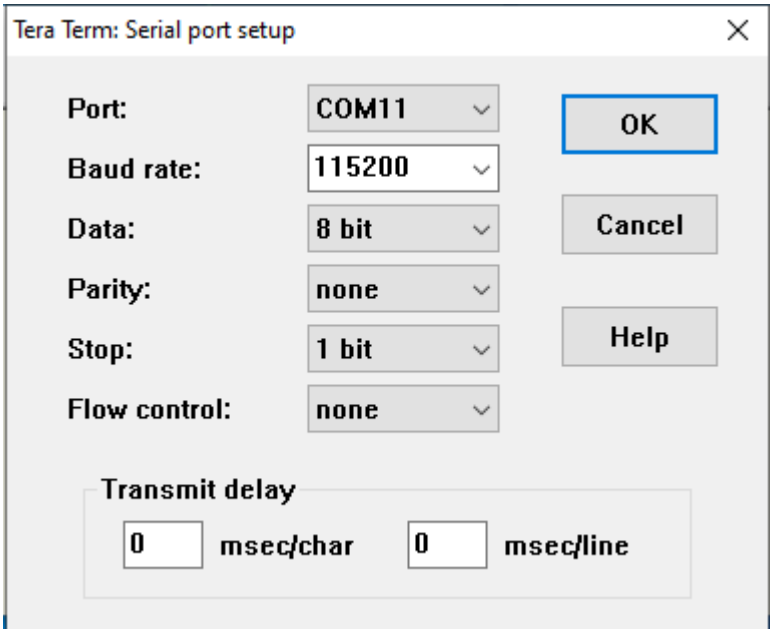
- One serial cable from the TAM station to the target PC.

- One Ethernet cable coming from NIC on the PC and going to the VLAN port of ethernet hub/switch

8.4.2.2 Loadset Preparation for 615A dataload

1. Ensure that target station is installed/loaded/configured to support COM11, COM5, CPA-2079 Discretes , TeraTerm, 615A dataloder before starting target execution. Refer <Verification-Branch>/tools/Target_Tools for the tools.
2. Open TeraTerm for P5_COM1 (Port=COM11) and P2_COM1 (Port=COM5), configured as shown below (depending on the test station configuration, COM11 and COM5 may be used instead).

P5_COM1:



P2_COM1:

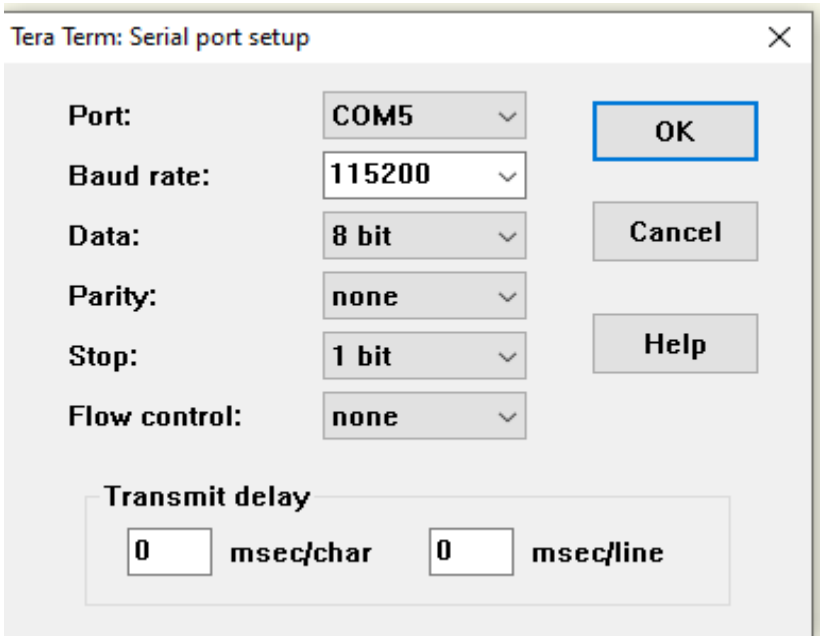


Figure 47: Tera Term:Serial port setup

3. Launch tam_disc.exe shortcut from desktop and configure as shown below.

TAM Discretes

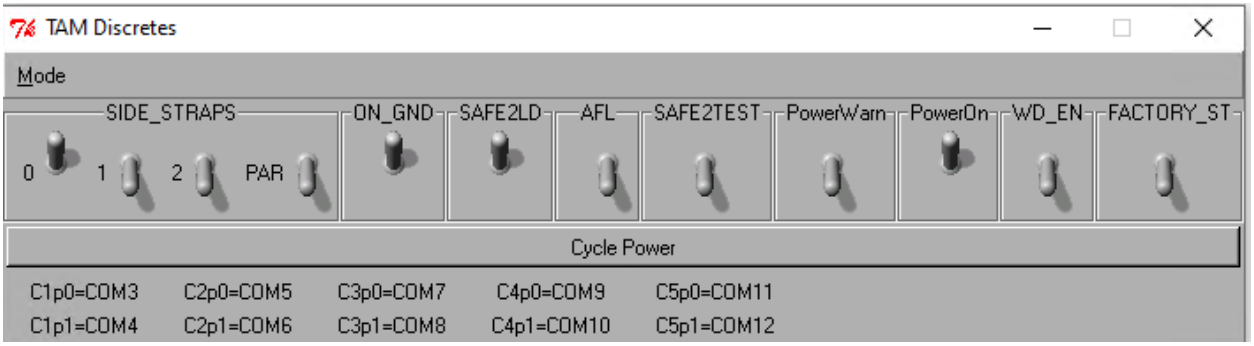
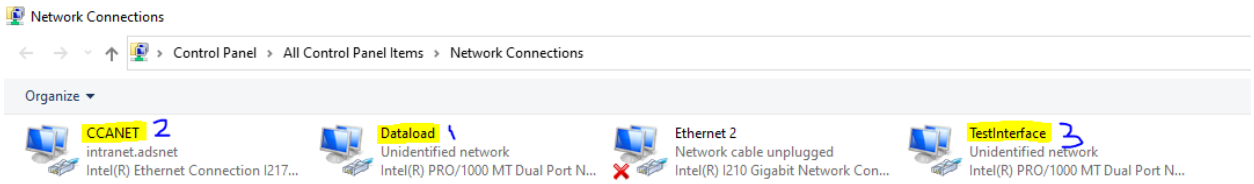


Figure 48: TAM Discretes

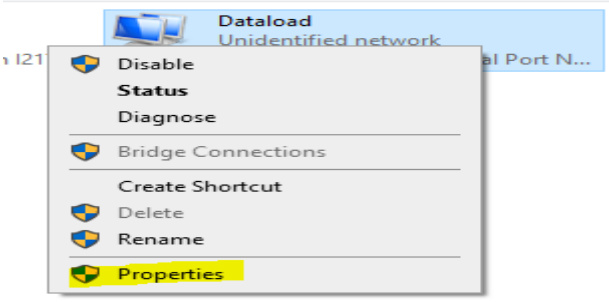
type in "CRV00160" without the quotes, and then click add, then click apply.

4. Set the'Interfacte Metrix' Value for Dataload Interface as '5' by performing the below steps, in EDS target pc.

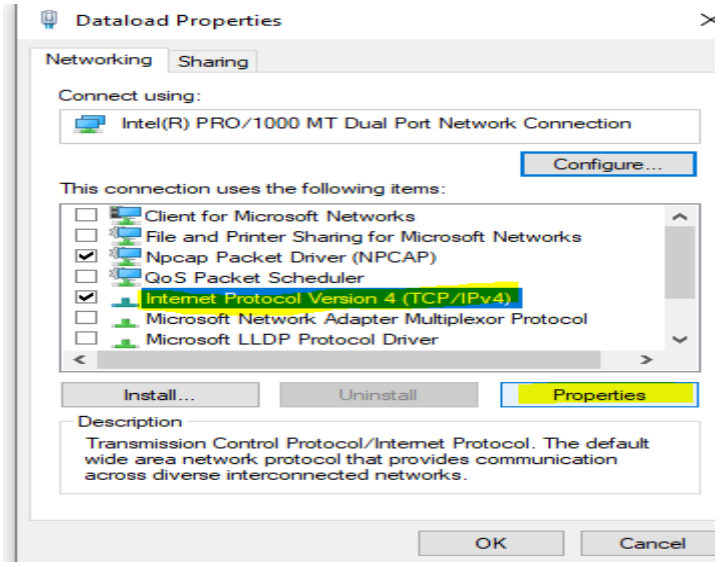
   4.1. Select 'Dataload' under Network Connections



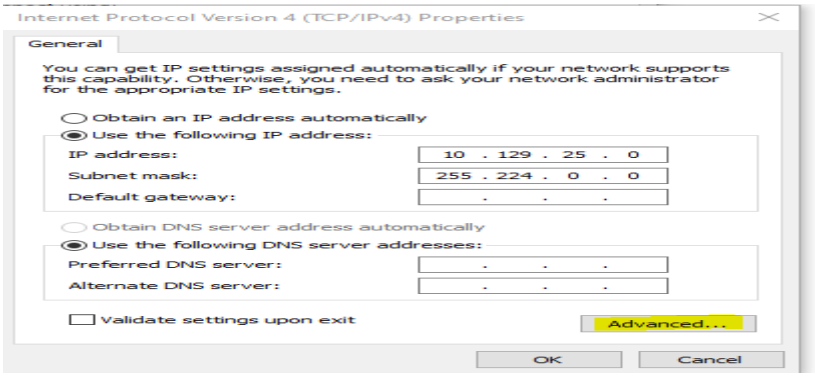
   4.2. Right click on the Dataload interface and select 'Properties'



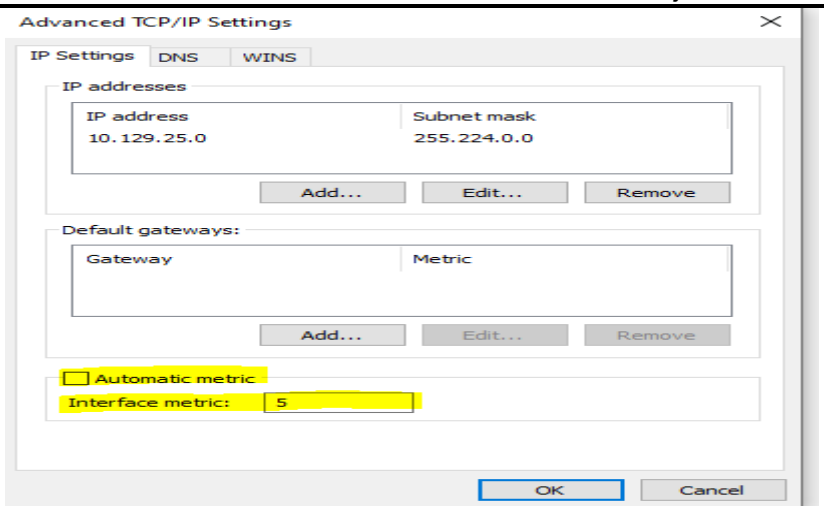
4.3. Select 'Properties' under the 'Internet Protocol Version 4(TCP/IPv4)



4.4. Select 'Advanced…' button



4.5. Uncheck the 'Automatic Metric' and Update the Interface metric value as '5'



5. Set 'Interface Metric' value for CCANET/Alliance Interface as '15' by following step 5.1 to 5.5 for CCANET interface.
6. Set 'Interface Metric' value for Test Interface as '10' by following step 5.1 to 5.5 for Test Interface.

8.4.2.3 TTL files for Target

TTL files are located at /vista_sim_EDS/scripts/

A_PM_CREATE.ttl

This is a teraterm macro file to send start process commands using teraterm communication port to LEFT PM.

A_PM_KILL.ttl

This is a teraterm macro file to send kill process commands using teraterm communication port to LEFT PM

L_DLCA_CREATE.ttl

This is a teraterm macro file to send start process command using teraterm communication port to LEFT DLCA.

L_DLCA_KILL.ttl

This is a teraterm macro file to send kill process command using teraterm communication port to LEFT DLCA

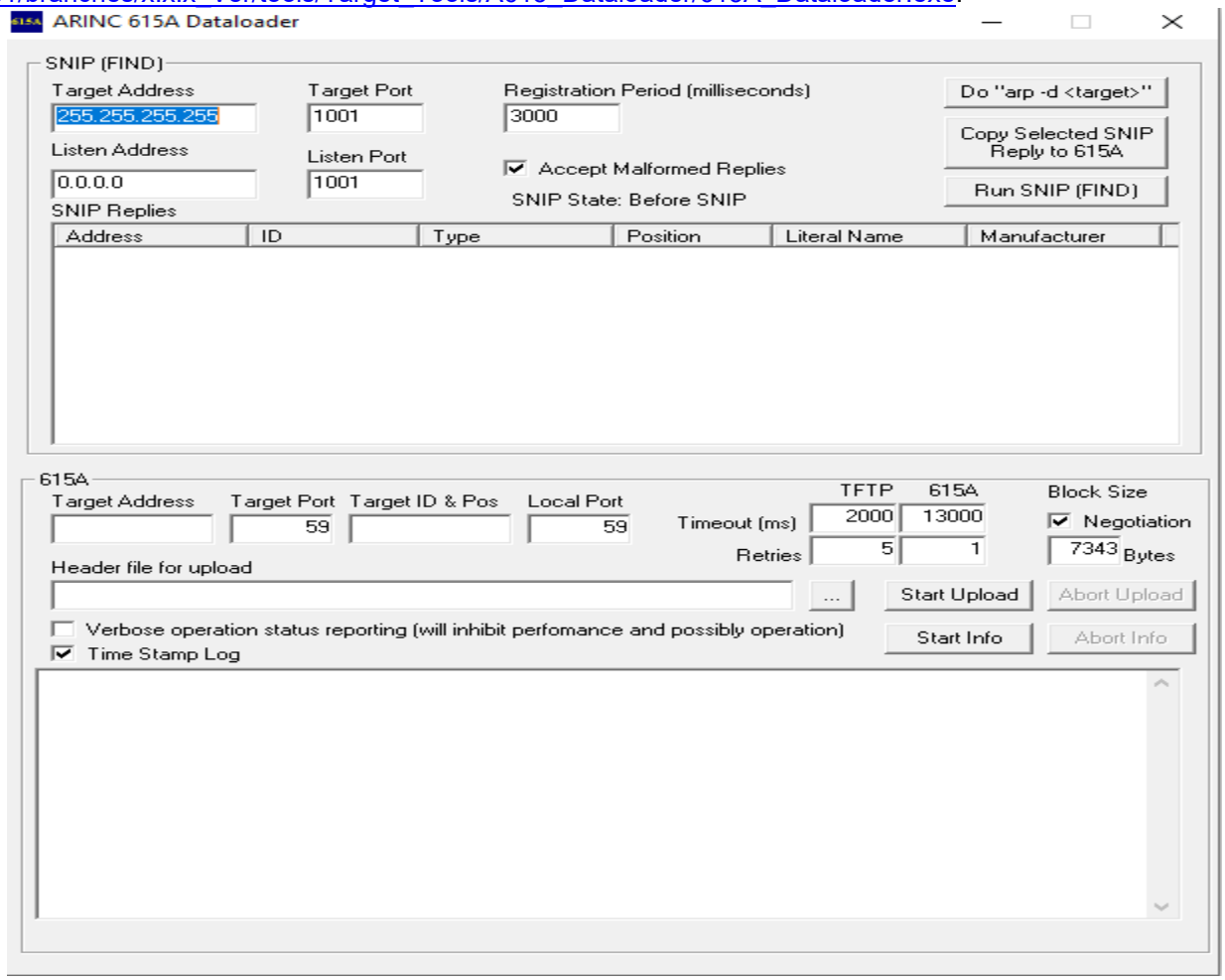
L_DLCA_NVM_PURGE.ttl

This is a teraterm macro file to send command nvm_purge using teraterm communication port to LEFT DLCA.

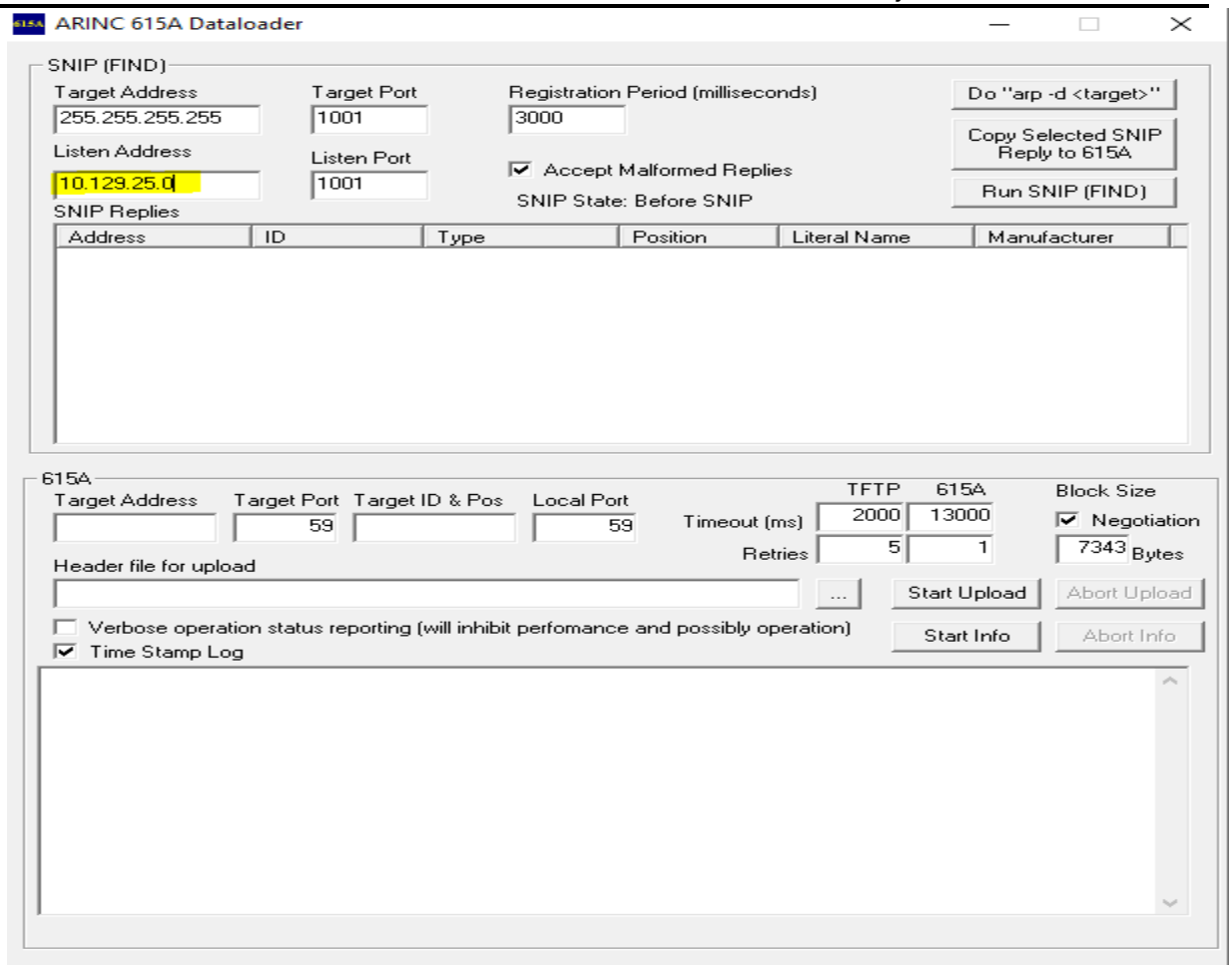
8.4.2.4 615A load

1. To perform a 615A load, a target build will need to have been successfully completed as per the guidance in the section 6.4 EDS Target Build, for EDS Target Build; and in the section 6.5 EDS Target Build with Invalid XML files, for EDS custom build. Copy the media set which need to be loaded into the EDS Target PC.
   Note: For dry-runs and RFS, get the latest builds from
   For normal build/custom build: <Dev-Branch>\Build\releases,
   For Timing build: <Verification-Branch>/test_procedures/TIMING/Mediasets/x.x.x
   where x.x.x is latest build version folder

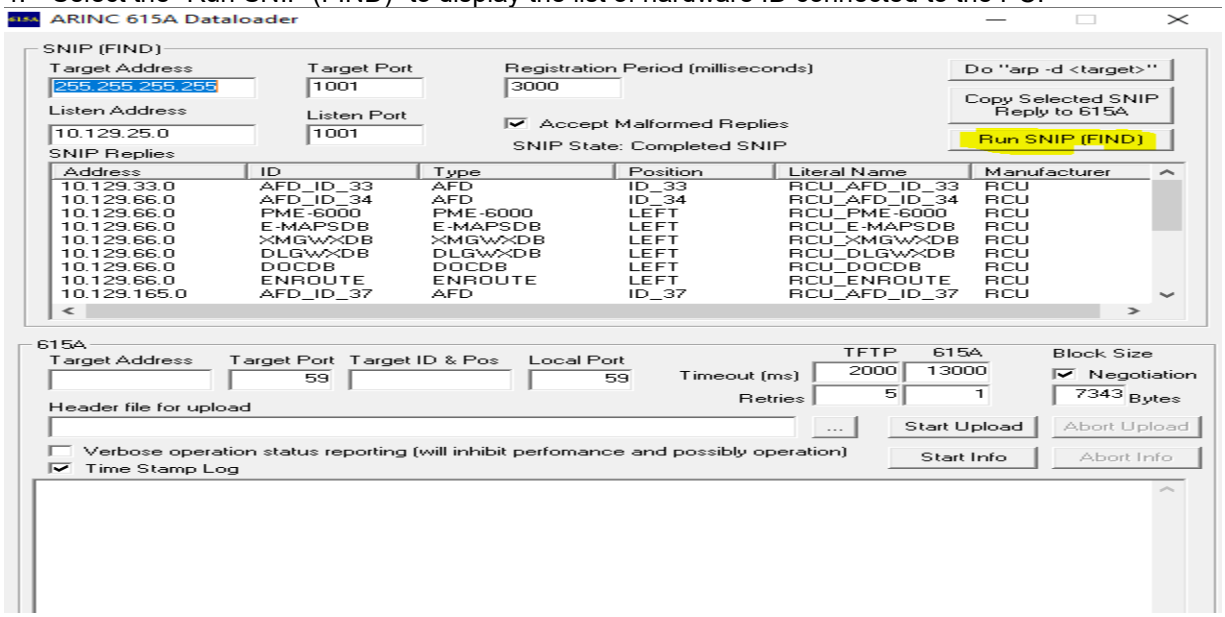
2. Start the 615A_dataloader.exe from the local drive repository path which has been checkedout from svn path,
https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/tools/Target_Tools/A615_Dataloader/615A_Dataloader.exe.



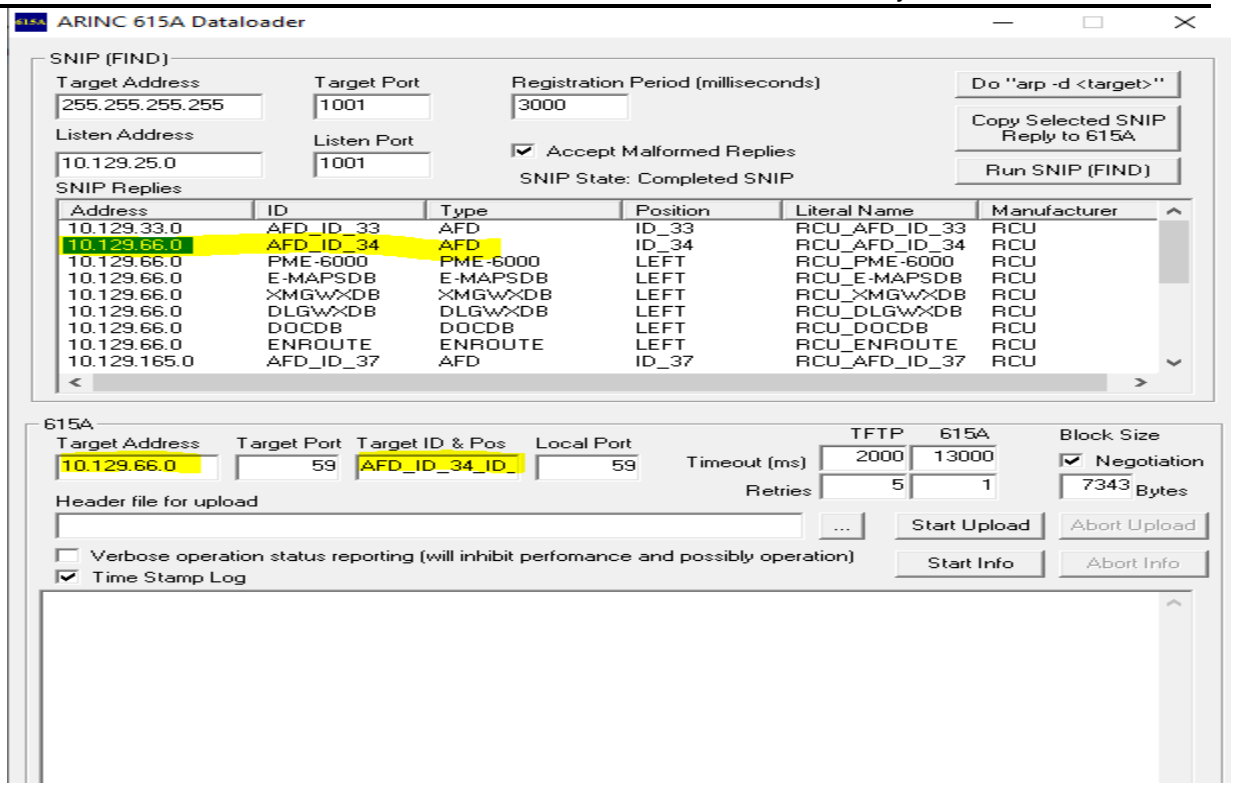
3. Update Listen Address as "10.129.25.0" (Dataload interface IP address).



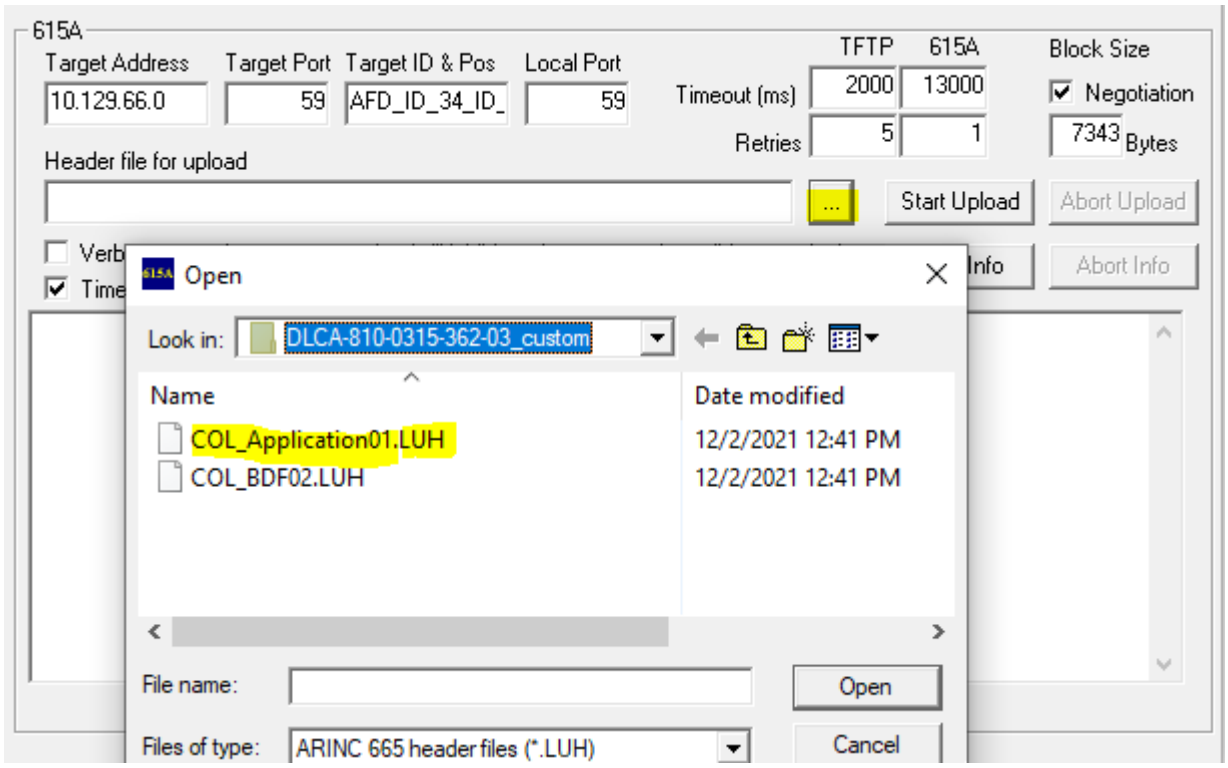
4. Select the "Run SNIP (FIND)" to display the list of hardware ID connected to the PC.



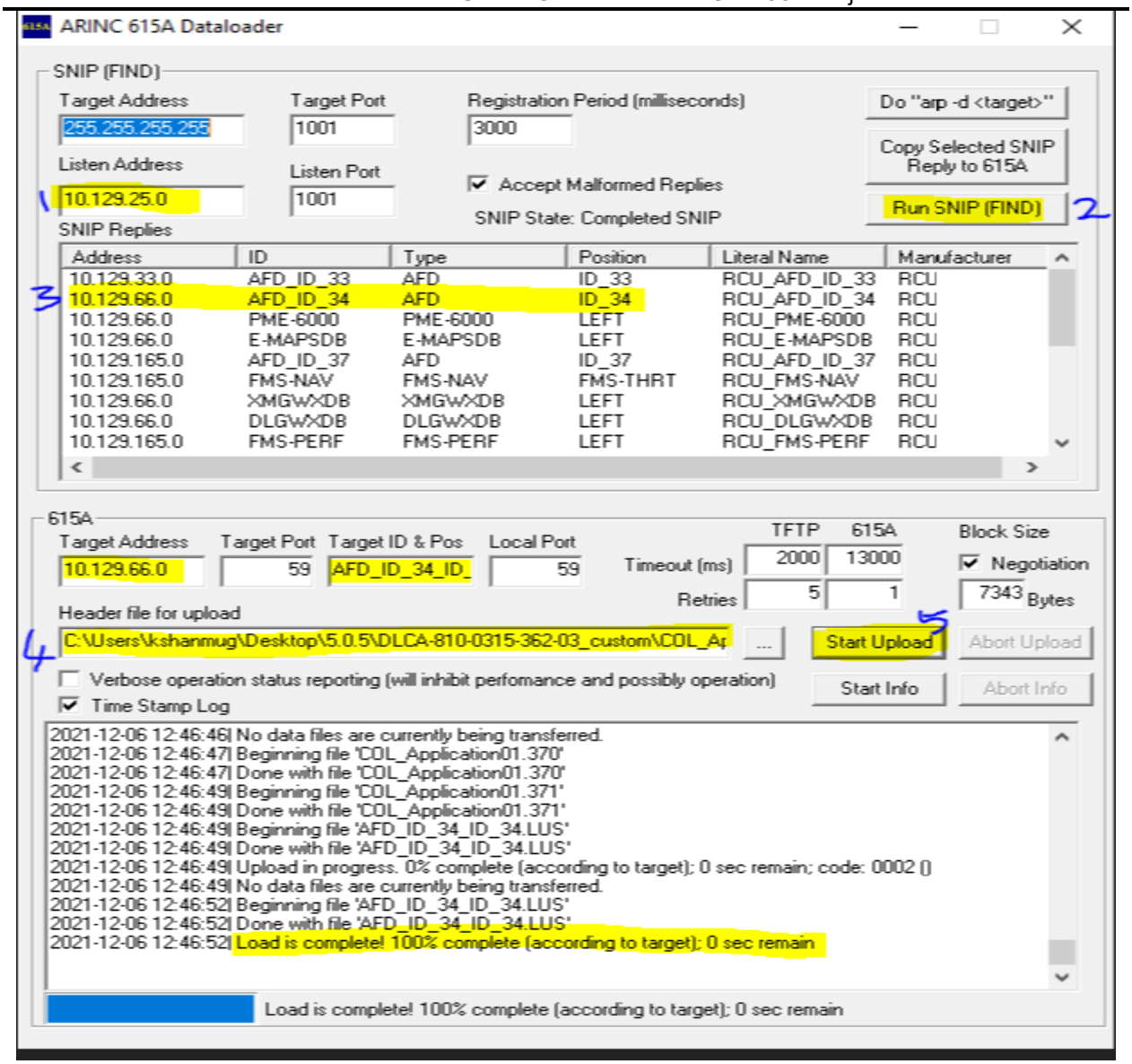
5. Select "AFD_ID_34" Id (Left AFD) from the listed hardware ID's name.



6. Select the header file which need to be loaded by selecting browse option.



7. And then press "Start Upload" to start the dataload operation,



8. Load is complete! 100% complete (according to target) message will be displayed when the dataload is completed fully.
9. Close CPA discrete window and Teraterm windows.

8.4.2.4.1 Starting Target

1. Start the VISTA environment in the https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/vista_sim_EDS/dlnkTarget.bat directory of SVN to start all of the hosted applications that connect to DLCA and PM.
2. Check for DLCA and PM Health Status to ensure DLCA is working. If it is off , Terminate the Vista environment and re-run A_PM_create.ttl and L_DLCA_create.ttl files
   [Refer to section 8.4.2.2 TTL files for Target, for svn path of the files]

8.4.2.5 USB load

USB load method is alternate method for loading on EDS Target

1. Copy all the media set files from the below svn path to the USB drive root path.

<Dev-Branch>/Build/EDS_Mediaset/MediaSet_CL604/

2. Replace folder "DLCA-810-0315-362-03" in the USB, with latest EDS build from <Dev-Branch>/Build/releases
3. Insert the USB drive into the EDS Target test station.
4. Set the discrete pins as per below for doing the dataload on Left DLCA,

   FL - UP
   STT - UP
   STL - UP
   OG - UP
   SSP - MIDDLE
   SS2 - MIDDLE
   SS1 - MIDDLE
   SS0 - UP

5. Login to the CRL27088 PC which is connected to the EDS Target station.
6. Start the TAM_discretes tool from the desktop.
7. Start the P1_Com1, P2_Com1, P3_Com3, P4_Com1 and P5_Com1 teraterm window from the desktop.
8. Power cycle the Target station (Either through power switch or through 'Power on' discrete from the TAM_discretes tool)
9. After the successful dataload complete "Safe to remove USB" message will be displayed in the AFD display.
10. Set the discrete pins as per the below after doing dataload,

    FL - MIDDLE
    STT - MIDDLE
    STL - MIDDLE OG - UP
    SSP - MIDDLE
    SS2 - MIDDLE
    SS1 - MIDDLE
    SS0 - UP

11. Run the below command in the Processor 2 teraterm(P2_Com1) window and make sure part numbers are correct or not.,

    /mnt/dlca/dlca.stripped

12. .Ensure DLCA is up and running, by following steps in section 8.4.2.3.1 Starting Target.

    Note: For doing Dataload on Right DLCA, follow steps from 1 to 12; but with discrete pin settings in step 4 as

    FL - UP
    STT - UP
    STL - UP
    OG - UP
    SSP - MIDDLE
    SS2 - MIDDLE

SS1 - UP

SS0 - MIDDLE

And discrete pin settings in step 10 as

FL - MIDDLE

STT - UP

STL - UP

OG - UP

SSP - MIDDLE

SS2 - MIDDLE

SS1 - UP

SS0 - MIDDLE

8.5 Target-based Test Station(IPS - Dual DLCA)

8.5.1 Folder/exe Locations

CPA Discrete - <Verification-Branch>/tools/Target_Tools/CPA_Discretes_CTA2079

Filter Driver (IPS Target Only) - <Verification-Branch>tools/Target_Tools/FilterDrivers/filterdriver/M170

Trace32: <Verification-Branch>/tools/Target_Tools/Trace32

8.5.2 Configuration Details

8.5.2.1 Physical Connection Details

To begin with, the following cables and equipment is required to set up the dual target system.

- Two serial cables coming from the FGPA ports on the stations (for use with the CPA Discretes utility)
- Four serial cables (two per station) for COM 1 and COM 2 on the stations going to the computers (referenced in this document as COM 6&7 and COM 4&2 on the PC)
- Two ethernet cables coming from LAN A ports on stations going into a ethernet hub/switch.
- Two ethernet cables coming from two NIC's on PC and going to the ethernet hub/switch.
- One ethernet cable to connect the Left and Right CTA stations.
- JTAGs

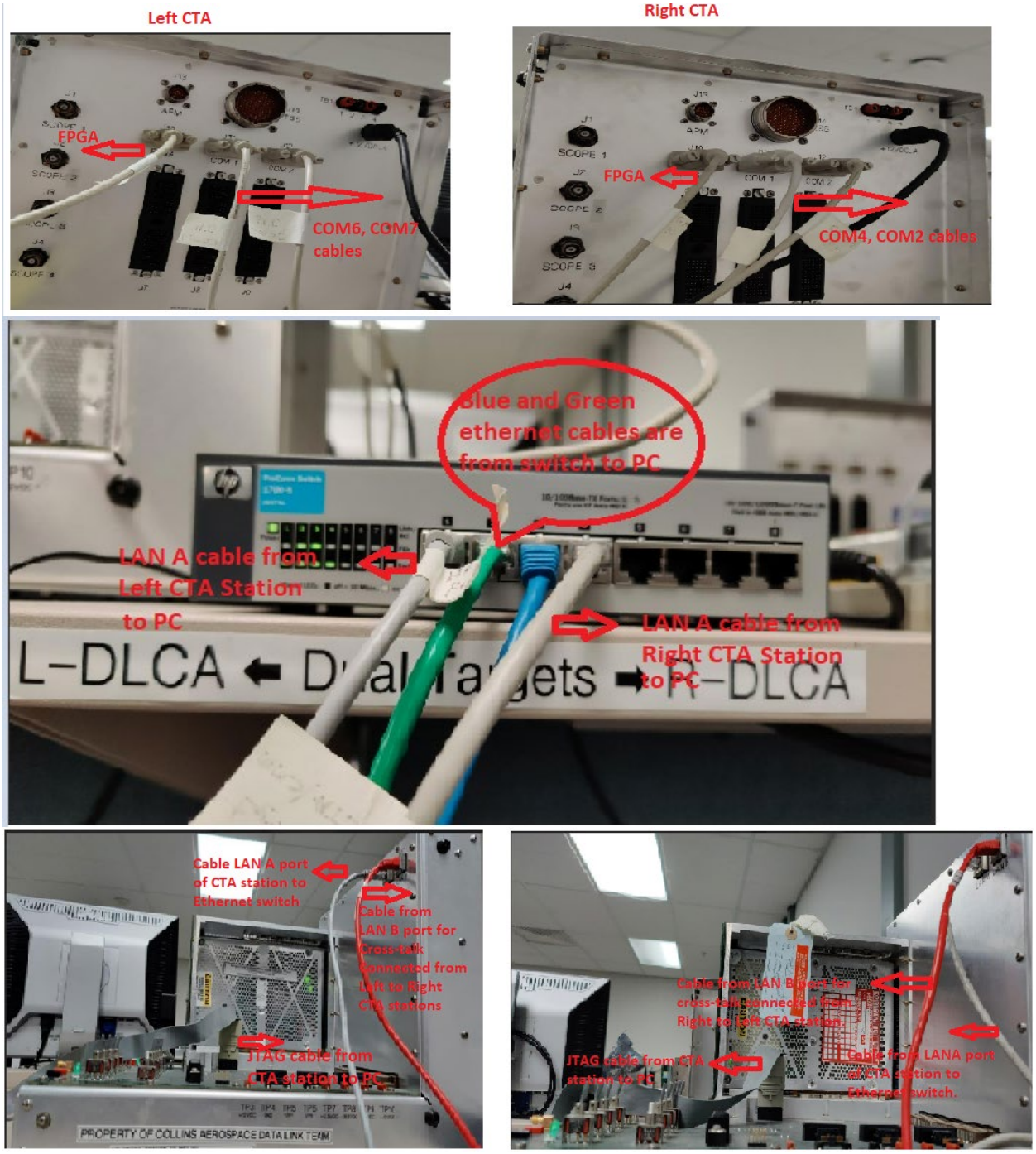
Connect up the six serial connections to the PC (most likely multiple serial to USB connectors will need to be used). The current COM setup referenced throughout the rest of this document is as follows:

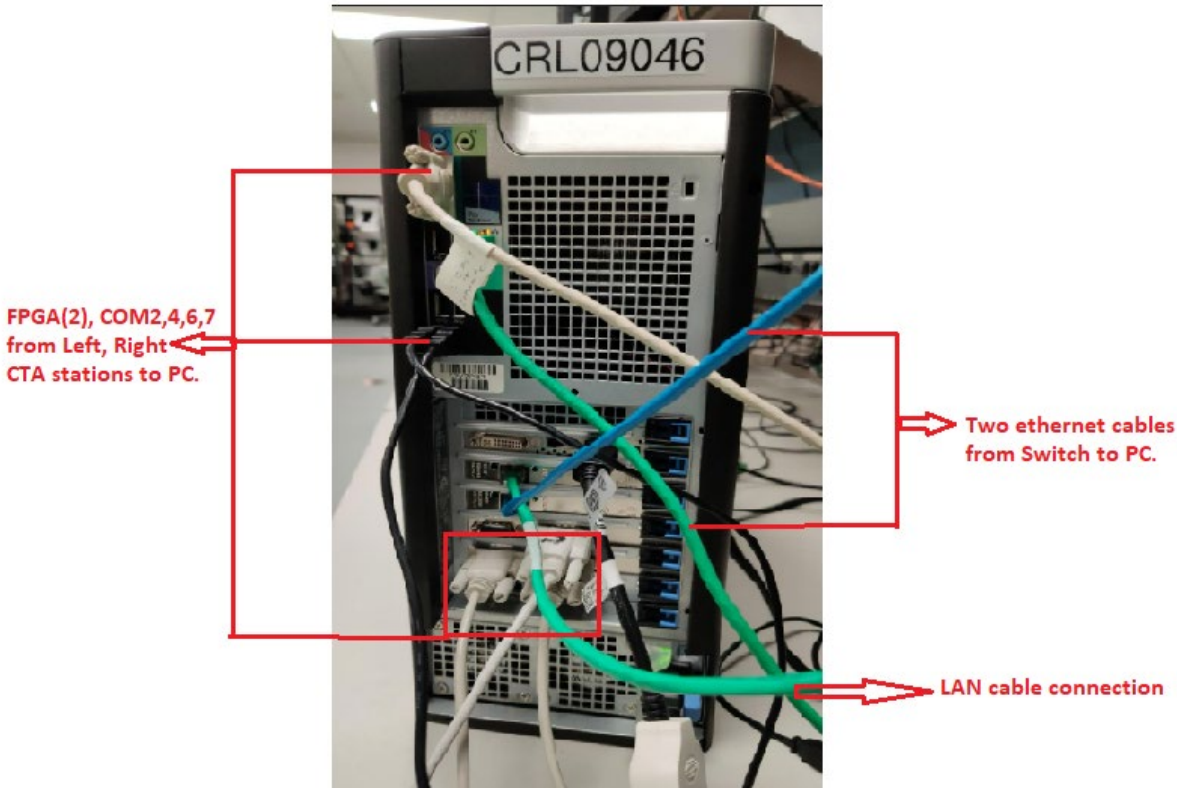
- COM 1 on the PC is the FGPA (for CPA Discretes) for the Left Side DLCA
- COM 5 on the PC is the FGPA (for CPA Discretes) for the Right Side DLCA
- COM 6 and COM 7 on the PC are the serial ports (COM 1 and COM 2 respectively) coming from the station for the Left Side PM and DLCA
- COM 4 and COM 2 on the PC are the serial ports (COM 1 and COM 2 respectively) coming from the station for the Right Side PM and DLCA
- Connect the ethernet cable coming from LAN A/LAN 1 port of Left CTA station to the ethernet hub/switch.
- Connect the ethernet cable from ethernet hub/switch to the target station.
- Connect the ethernet cable coming from LAN A/LAN 1 port of Right CTA station to the ethernet hub/switch.
- Connect the ethernet cable from ethernet hub/switch to the target station.
- Connect the JTAG from the Left CTA station to the target PC

- Connect the JTAG from the Right CTA station to the target PC

Cross-talk connection: Connect the ethernet cable from 'LANB' port of Left CTA station to 'LANB' port of Right CTA station.

Refer the below images for connection details:

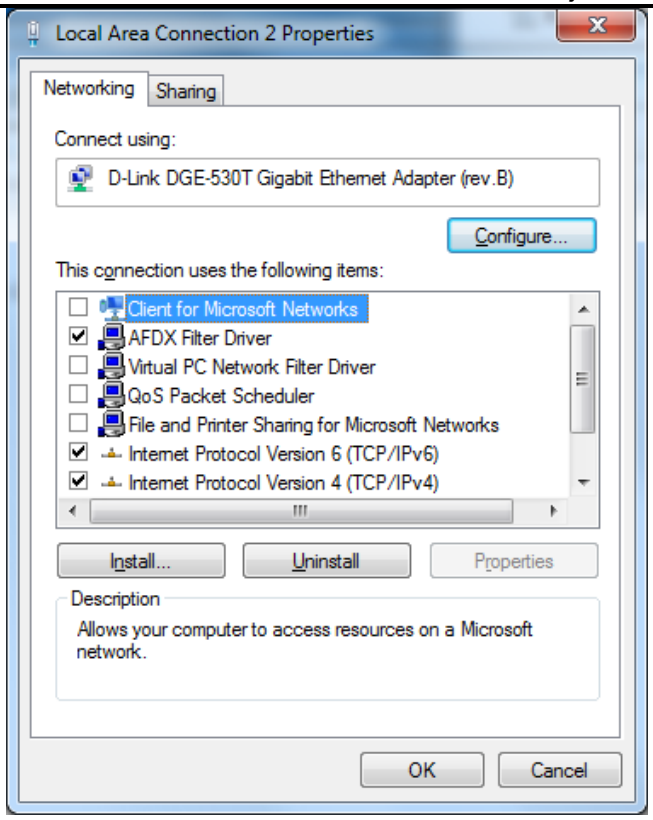




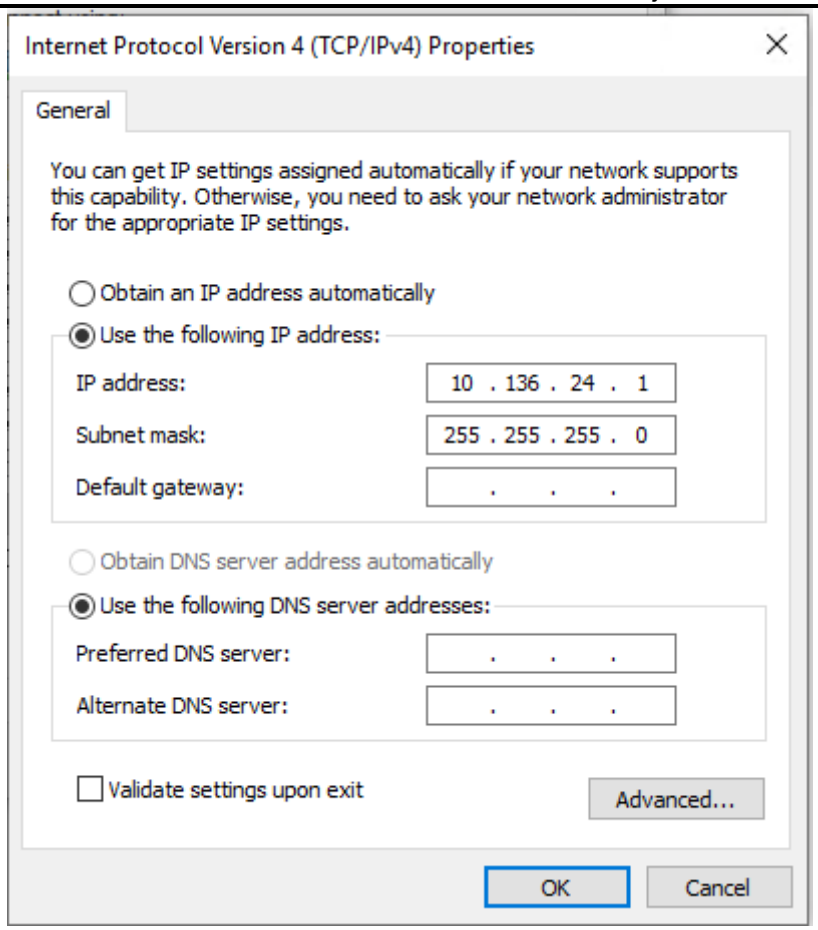
8.5.2.2 Configuring AFDX Network

1. Copy the setup.exe file to C:\rw_apps and execute the exe.
2. Launch the AFDX Filter Driver Controller short cut from desktop
3. Select the correct LAN connection and select LAN A
4. Browse the DLCAfilterdriver.bin file from SVN and load the file. select Activate button to install the driver

5. In My computer, Network connection select properties and uncheck all options except the ones shown below

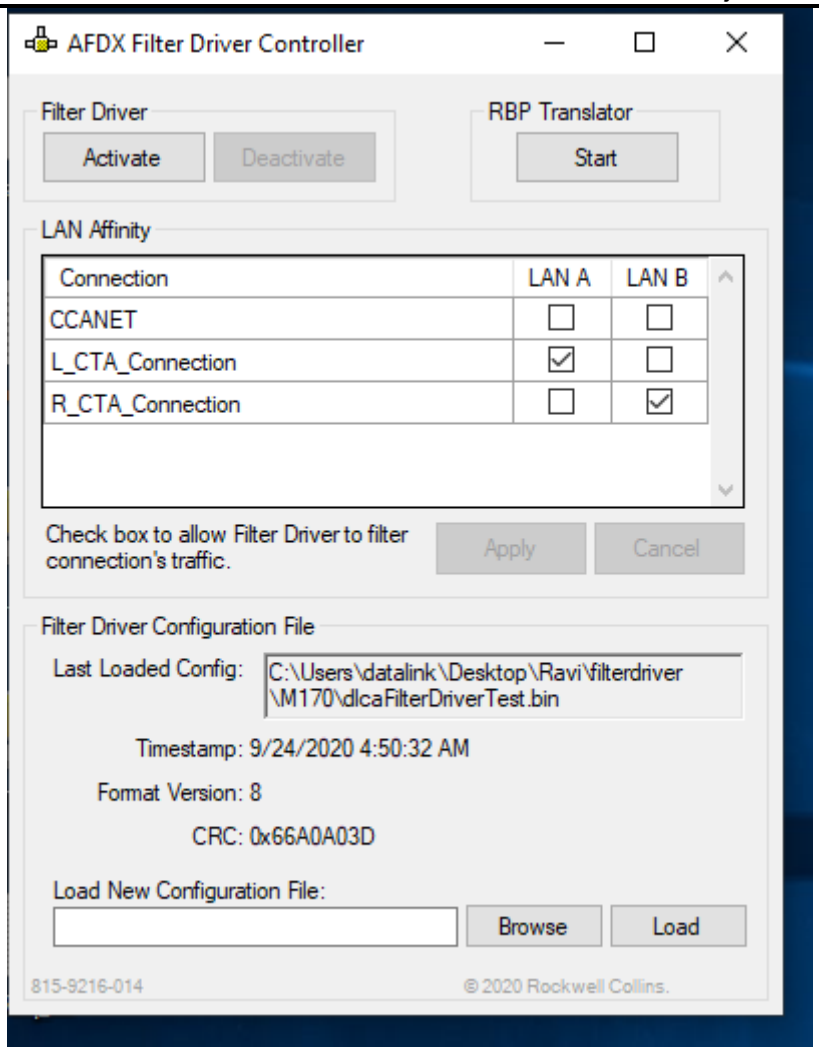


6. In TCP/IP4, select properties and set the IP address as shown below



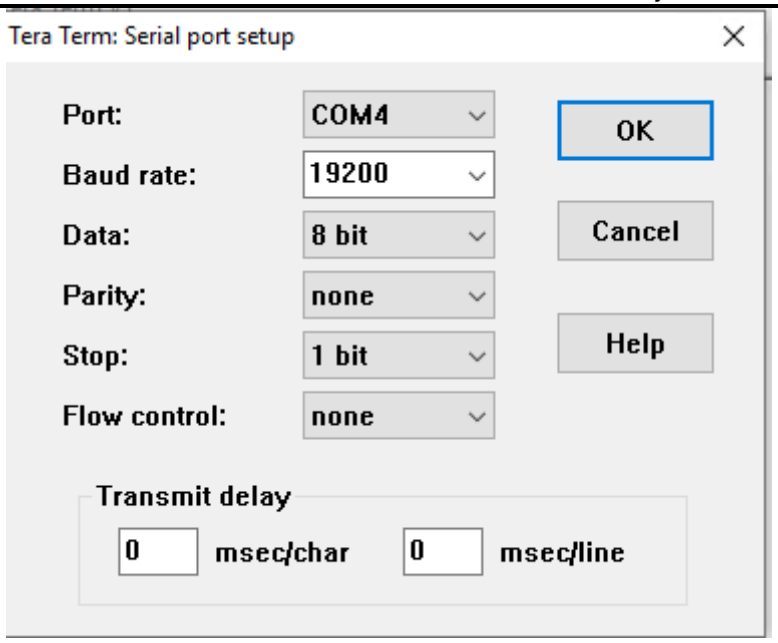
Open up the properties for each NIC cards. The IP addresses need to be set so that the LAN A card is 10.136.24.1, and LAN B is 10.136.25.1.

Ensure that the correct Filter Driver configuration is loaded (e.g. http://asvn/dlnk-dlca/branches/1.13.1_dev/Build/M170/NCT/FilterDriver.bin). There should be two filter driver cards listed in the AFDX Filter Driver Controller utility, one marked for LAN A and the other for LAN B.

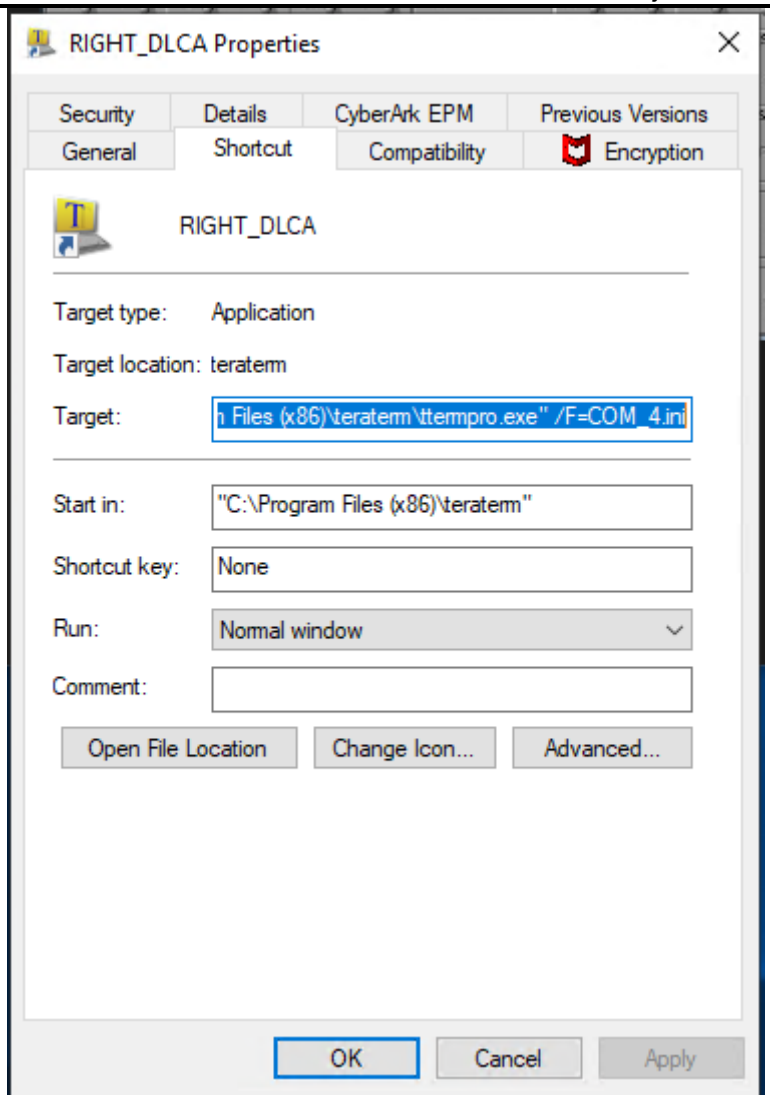


8.5.2.3 Configuring the Tera Term files

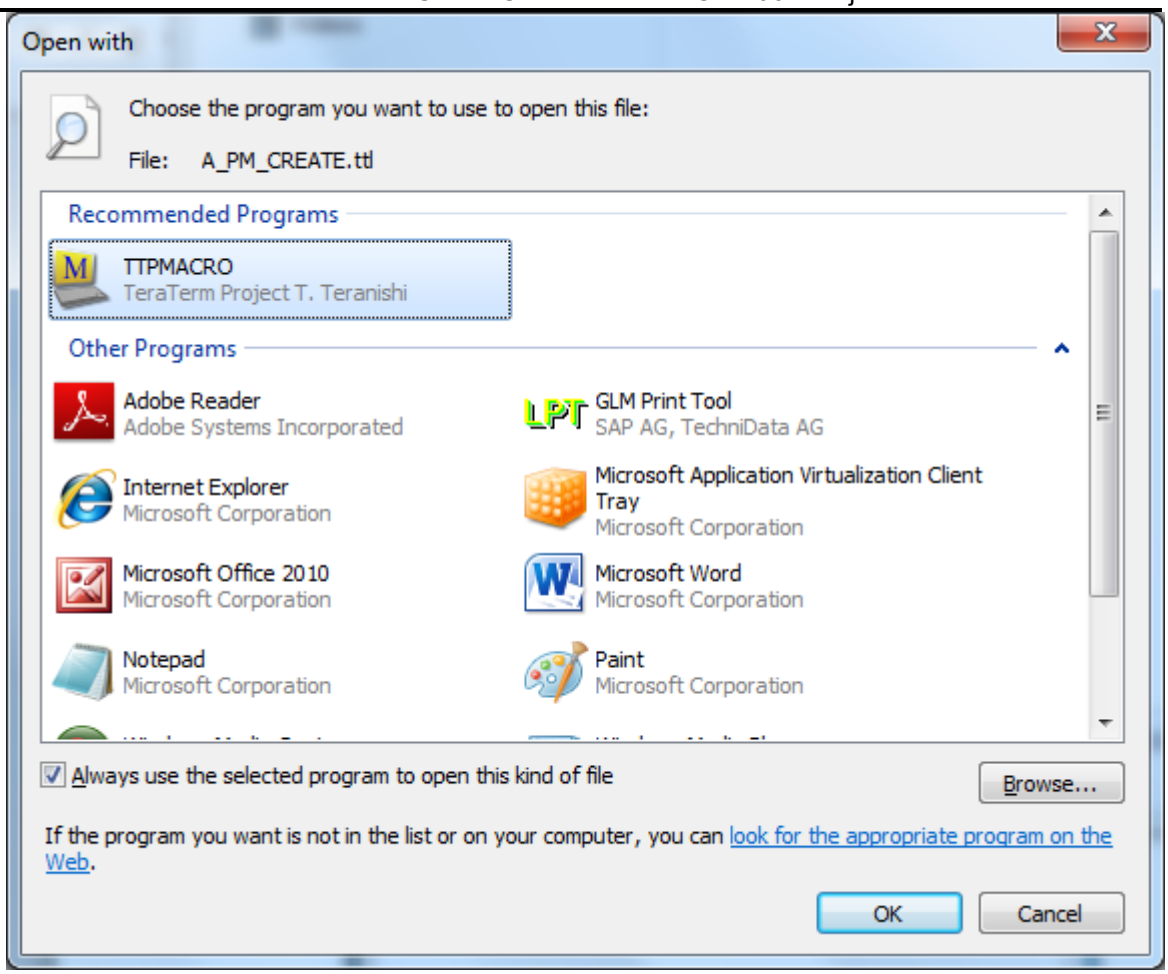
1. Copy the teraterm folder to C:\Program Files (x86)
2. Create 2 desktop short cuts to C:\Program Files (x86)\teraterm\ttermpro
3. Open the Tera Term short cut, in setup change the serial port settings as shown below



4. Similarly change for COM4.
5. In Desktop rename the shortcuts as RIGHT_DLCA and RIGHT_PM.
6. Right click on the RIGHT_DLCAshort cut and select properties
7. Set the INI file in target path as shown below



8. Similarly change for COM4.
9. For Verification, open folder https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/vista_sim_IPSDual/scripts
10. Open A_PM_CREATE.ttl file. Browse to path C:\Program Files (x86)\teraterm\ttpmacro and select check box "always open the program"

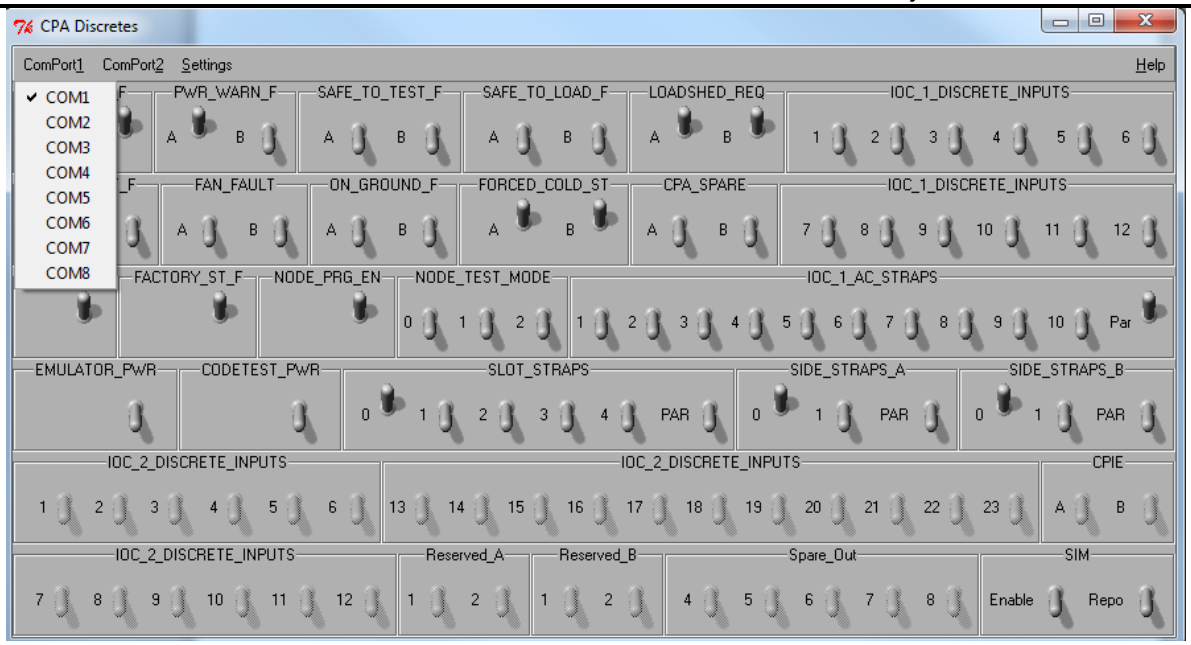


COM PORT NAME	APPLICATION
COM_6	Left PM
COM_7	Left DLCA
COM_4	Right PM
COM_2	Right DLCA

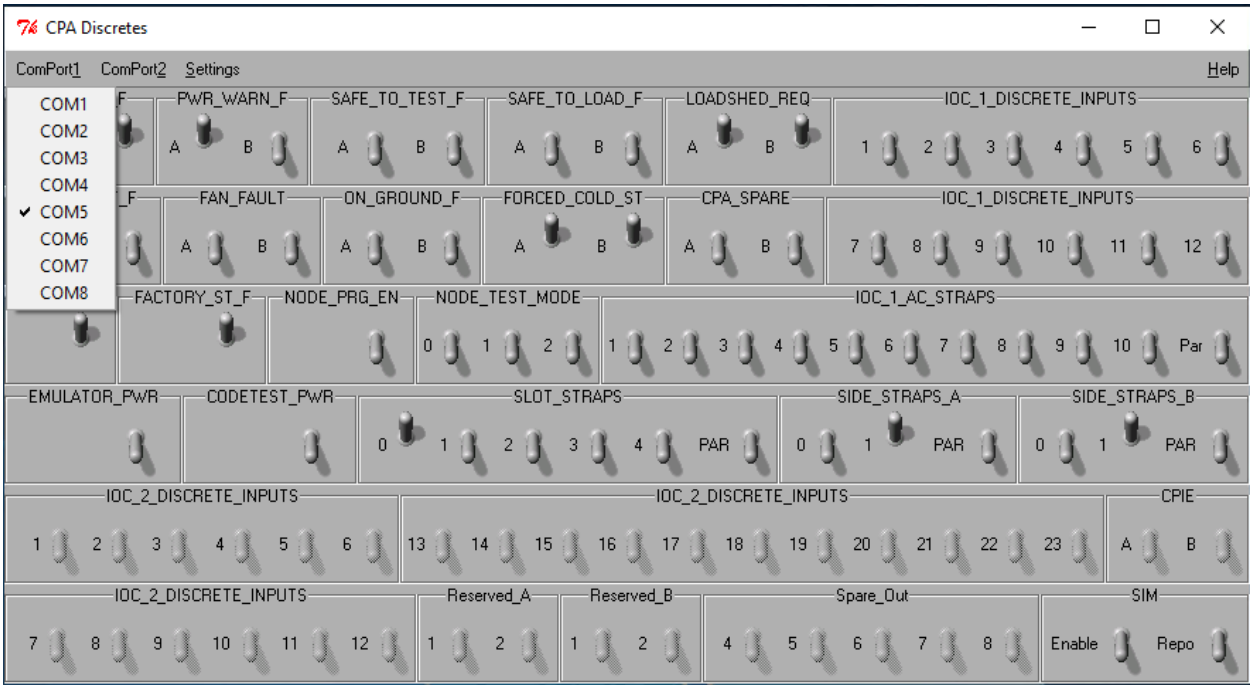
Note: The COM ports are configured locally on PC CRL09046. COM ports may be different in other machines.

8.5.2.4 CPA_DISCRETE:

1. Copy the CPA_Discretes_CTA2079 folder to C:\rw_apps.
2. Create a desktop shortcut to C:\rw_apps\CPA_Discretes_CTA2079\bin\cpa_disc_cta2079
3. For Left DLCA, Select ComPort1 → COM1 and do the below selections.



4. For Right DLCA, Select ComPort1 → COM5 and do the below selections



8.5.2.5 TTL files for Target

A_PM_CREATE.ttl

This is a teraterm macro file to send start process commands using teraterm communication port to LEFT PM.

A_PM_KILL.ttl

This is a teraterm macro file to send kill process commands using teraterm communication port to LEFT PM.

B_PM_CREATE.ttl

This is a teraterm macro file to send start process commands using teraterm communication port to RIGHT PM.

B_PM_KILL.ttl

This is a teraterm macro file to send kill process commands using teraterm communication port to RIGHT PM.

L_DLCA_CREATE.ttl

This is a teraterm macro file to send start process command using teraterm communication port to LEFT DLCA.

L_DLCA_KILL.ttl

This is a teraterm macro file to send kill process command using teraterm communication port to LEFT DLCA.

L_DLCA_NVM_PURGE.ttl

This is a teraterm macro file to send command nvm_purge using teraterm communication port to LEFT DLCA.

R_DLCA_CREATE.ttl

This is a teraterm macro file to send start process command using teraterm communication port to RIGHT DLCA.

R_DLCA_KILL.ttl

This is a teraterm macro file to send kill process command using teraterm communication port to RIGHT DLCA..

R_DLCA_NVM_PURGE.ttl

This is a teraterm macro file to send command nvm_purge using teraterm communication port to RIGHT DLCA.

8.5.3 Lauterbach Load

8.5.3.1 Left DLCA

1 To perform a Lauterbach load, a target build will need to have been successfully completed as per the guidance in the section 'Custom Builds Instructions', sub section IPS Target Build .
2 Create a folder in the target pc, and place localfs_<IPS_program name> generated as per section 6.1 IPS Target Build, for IPS Target Build. Similarly refer section 6.2 IPS Custom Builds for IPS Custom Builds , section 6.3 Timing Build, for Timing Build. For example create folder as U:\CSeries_dlca\ and place "localfs_cseries" build file for IPS Target Build.
Note: For dry-runs and RFS, get the latest builds from
For normal build: <Dev-Branch>\Build\releases,
For custom build: <Verification-Branch>/Reference/Custom_Build_Support/Builds/x.x.x
For Timing build: <Verification-Branch>/test_procedures/TIMING/Mediasets/x.x.x
where x.x.x is latest build version folder
3 Open two TeraTerm terminals, one for each of the VM's on the load. Typically PM and HM will output to COM6 and DLCA will output to COM7.
4 Open CPA_Disc.exe, as shown and ensure that the ComPort is select as COM1.

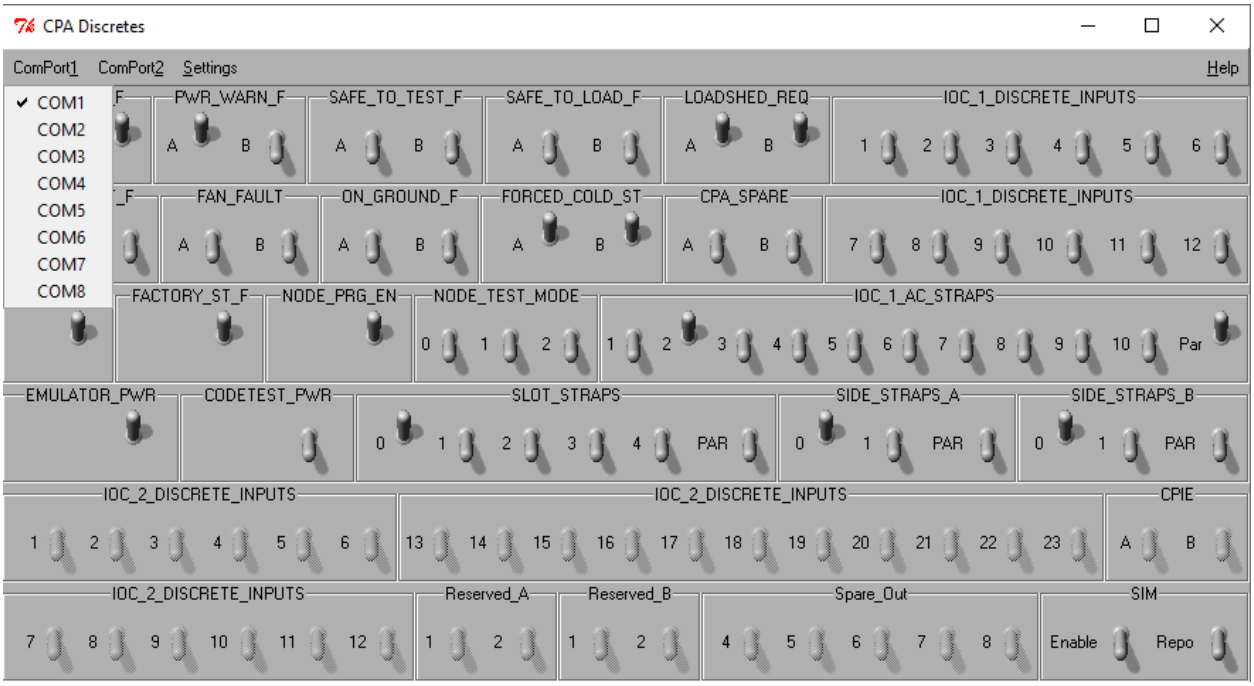
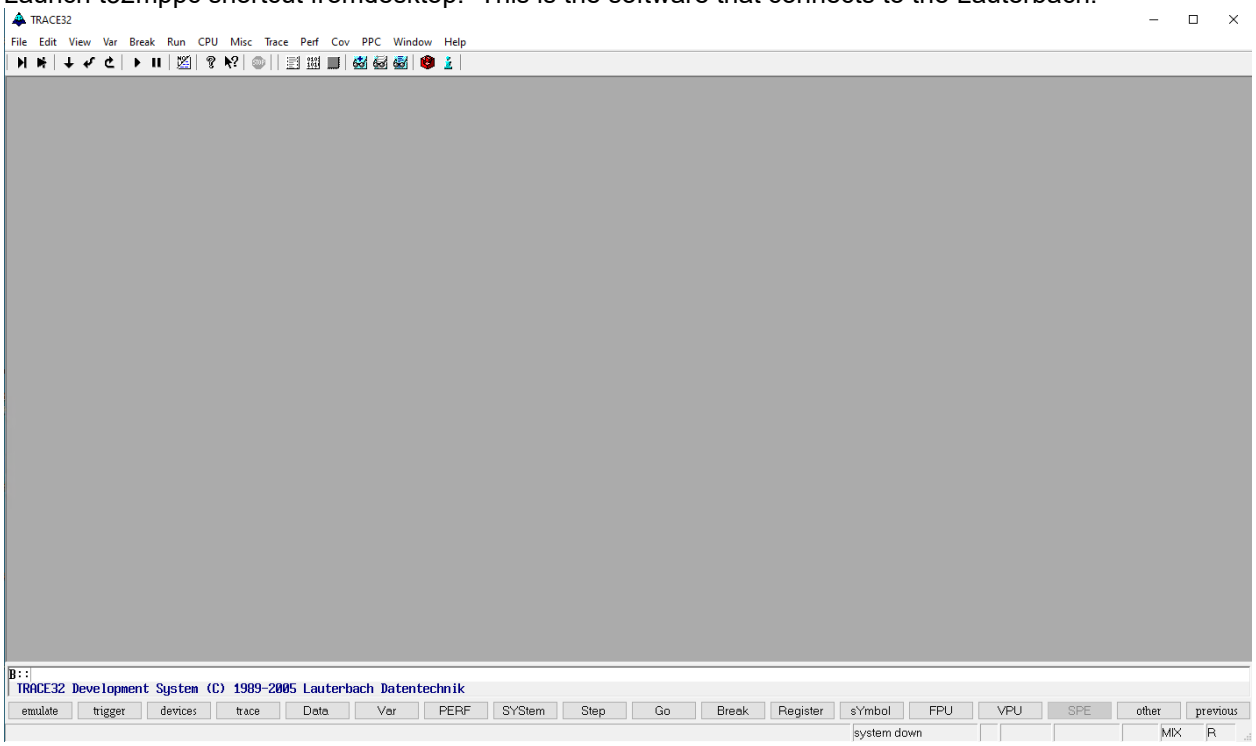


Figure 49: CPA Discretes

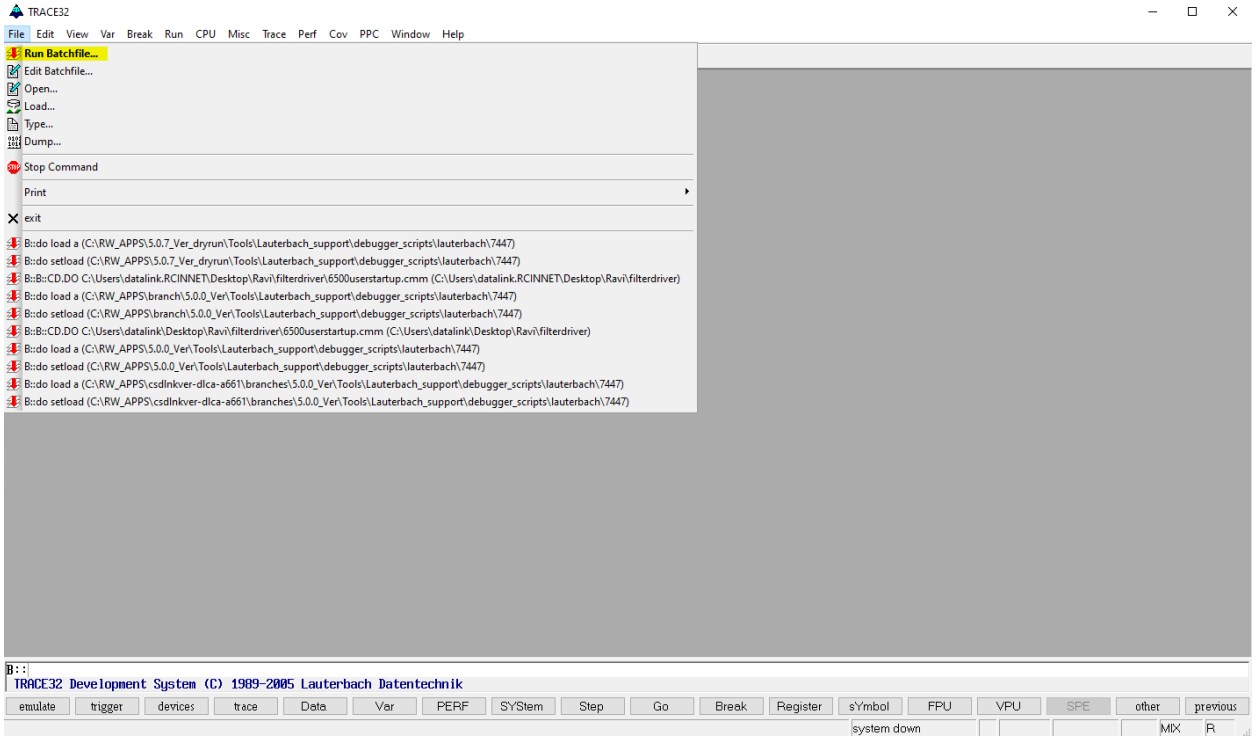
5 Ensure that Turn watchdog (WD_INH_F) is on.

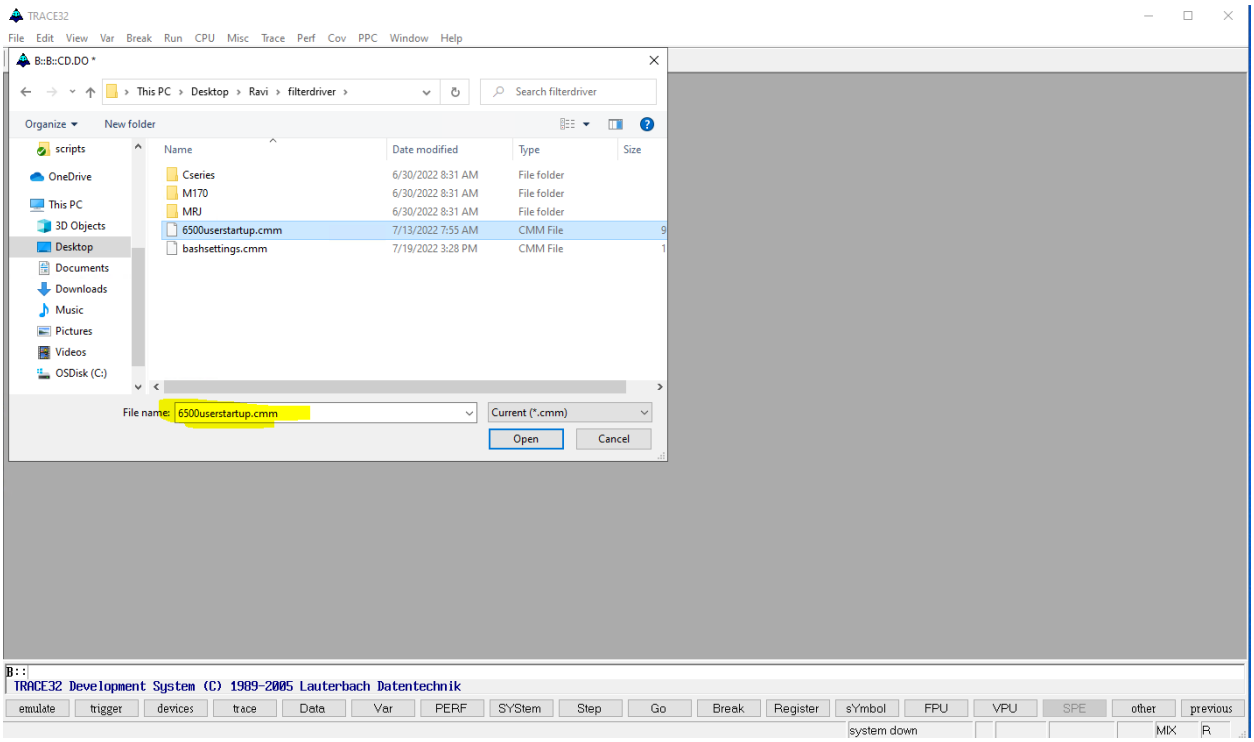
6 AFDX Filter Driver : Follow the steps as per the section " Configuring AFDX Network"

7 Launch t32mppc shortcut fromdesktop. This is the software that connects to the Lauterbach.

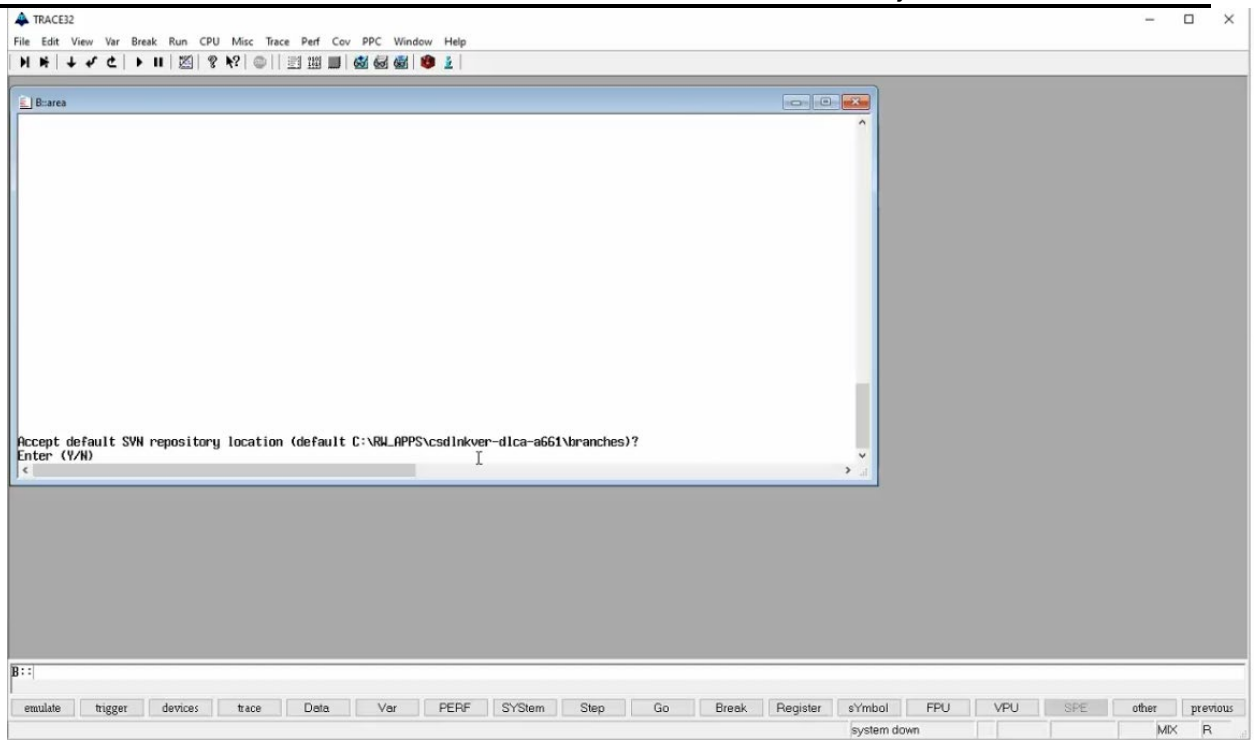


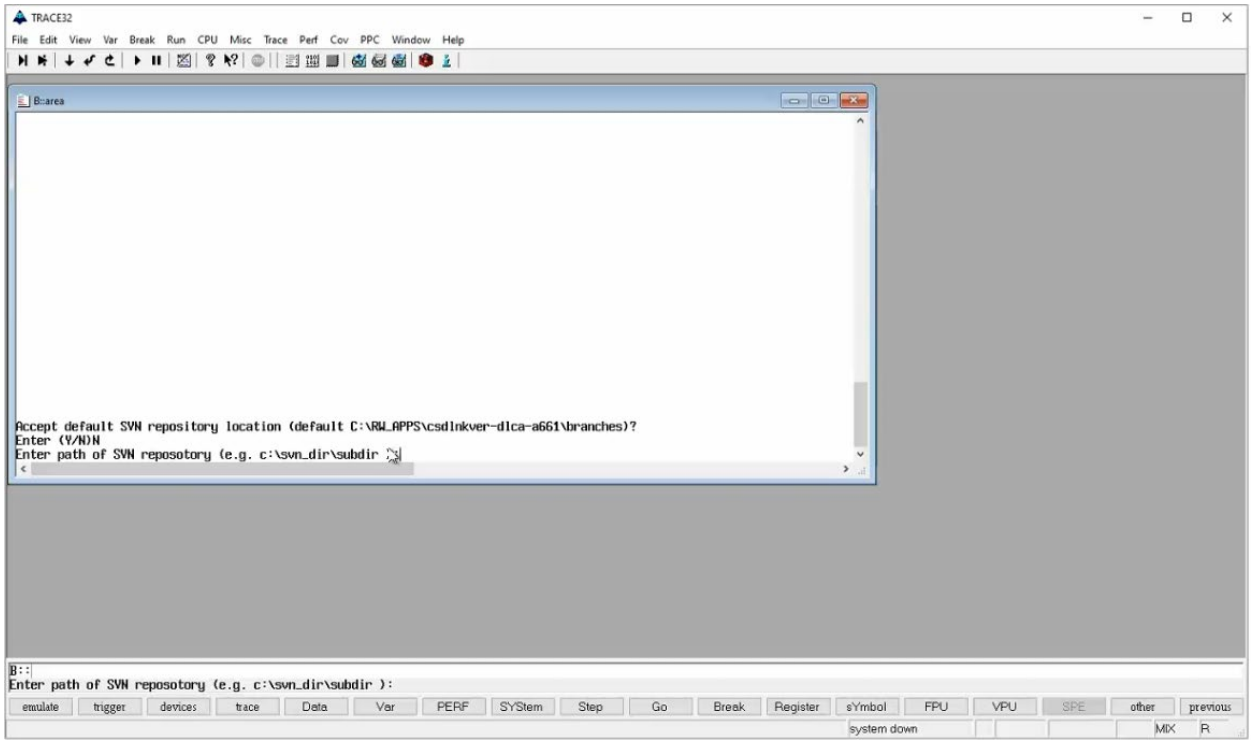
8 Run the "6500Userstartup.cmm" file from File->Run Batchfile

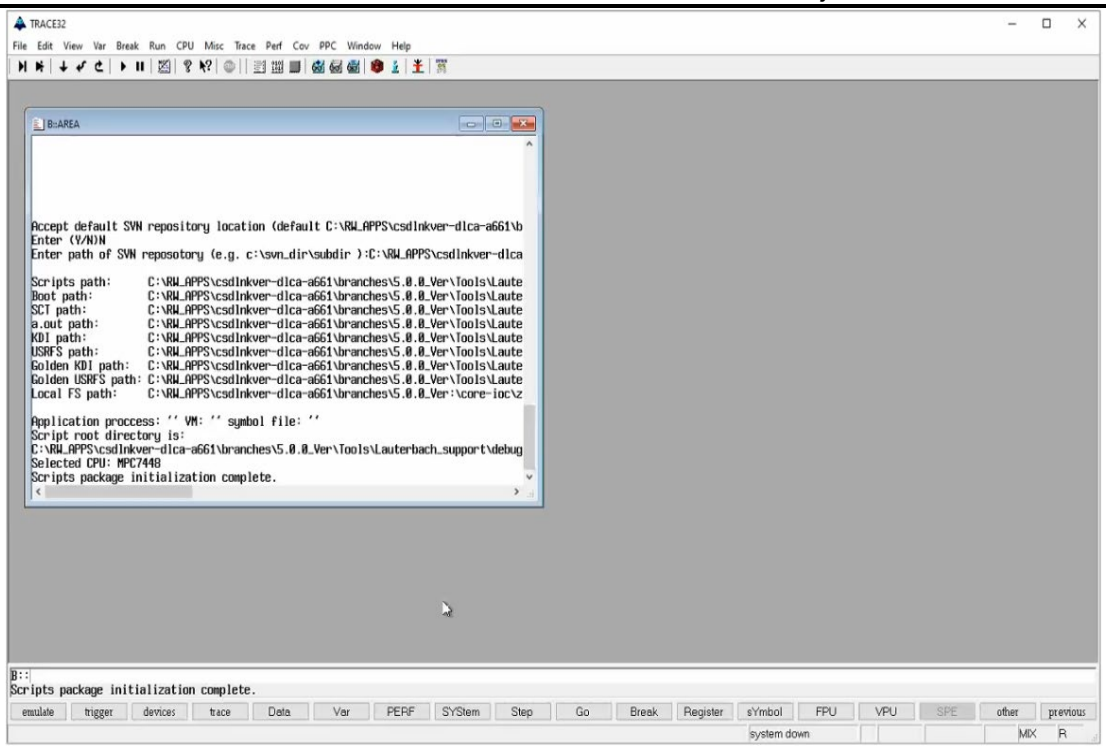




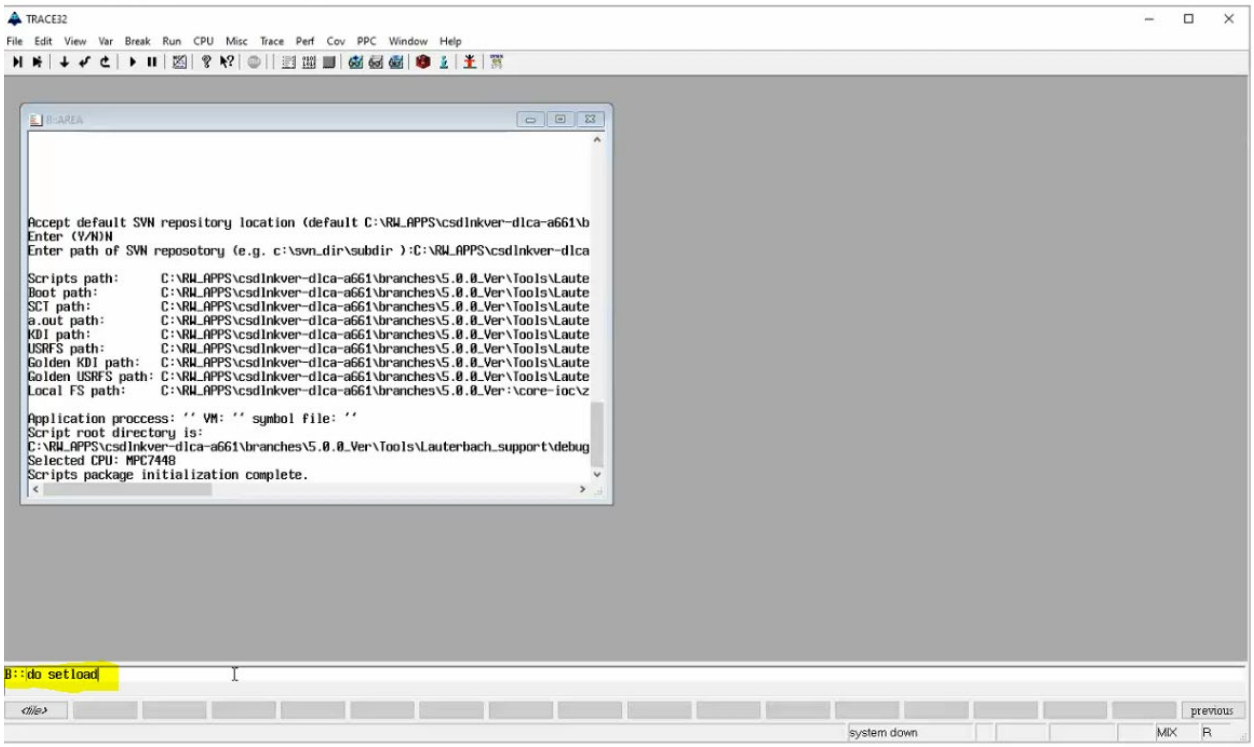
9 Enter 'Y' if the default SVN repository location shown is correct or Enter 'N' and Enter path of SVN repository as <Verification-Branch>





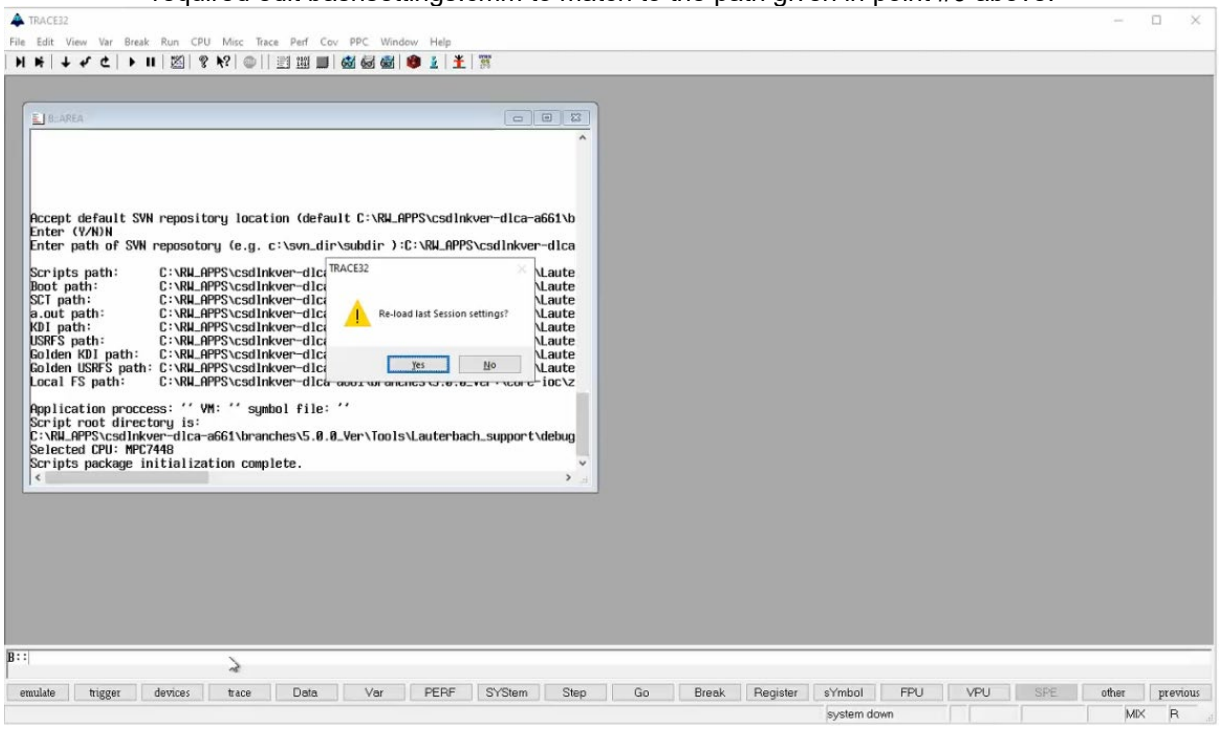


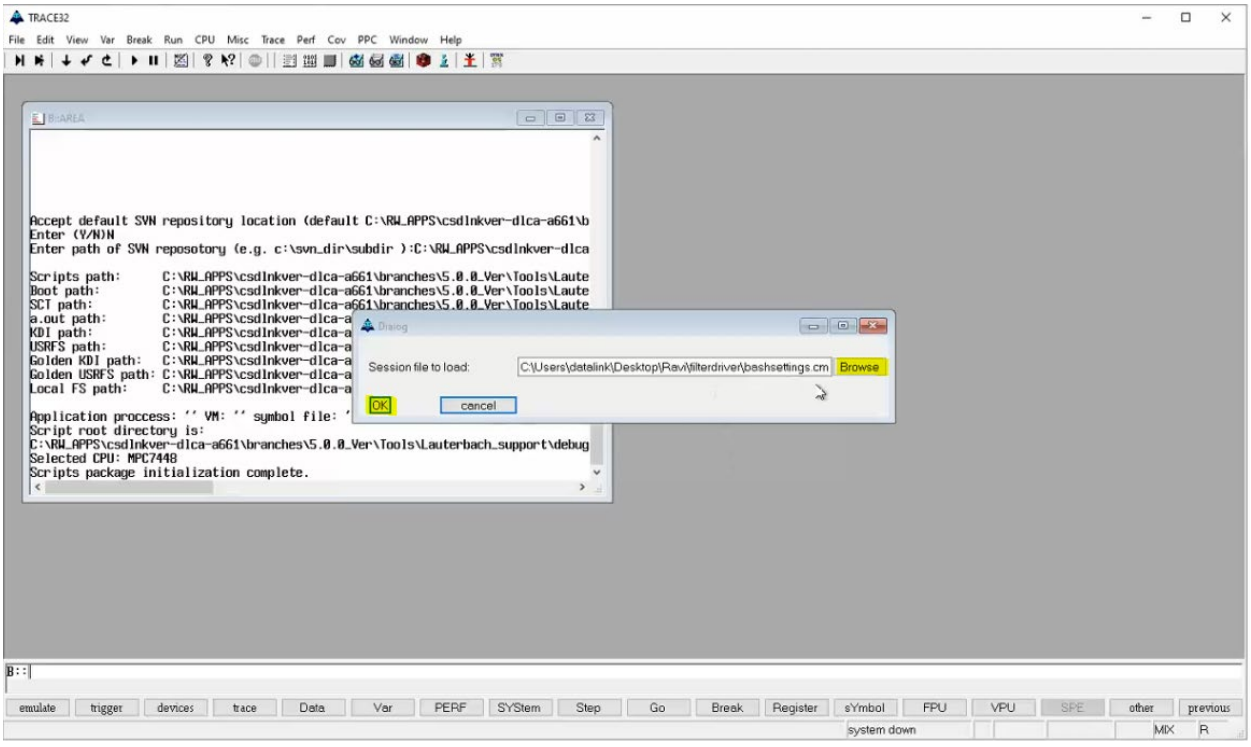
10 After seeing message, 'Scripts package initialization complete', Execute 'do setload' from the Trace32 command prompt. .



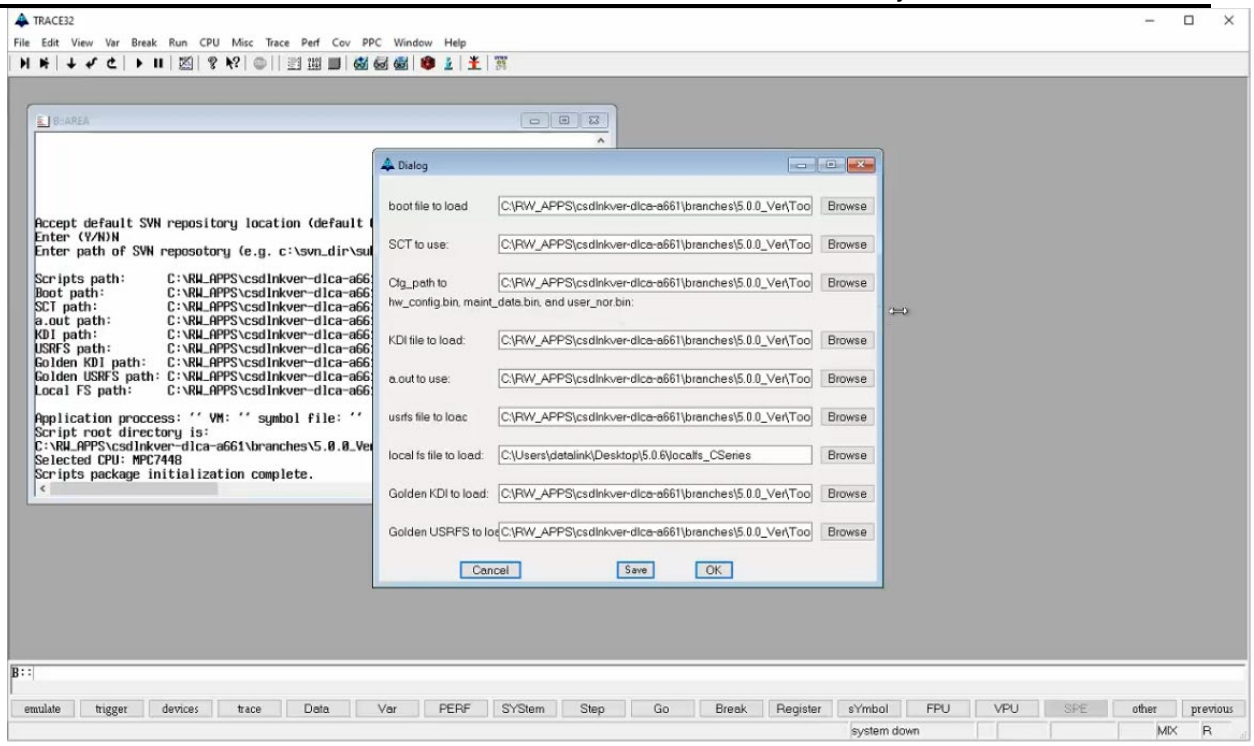
11 Choose 'Yes' to the Reload last Session settings popup, and browse to the setttings file bashsettings.cmm[checkedout from <Verification-Branch> /tools/Target_Tools/FilterDrivers/filterdriver/bashsettings.cmm], and select OK. Before selecting OK, if

required edit bashsettings.cmm to match to the path given in point #9 above.

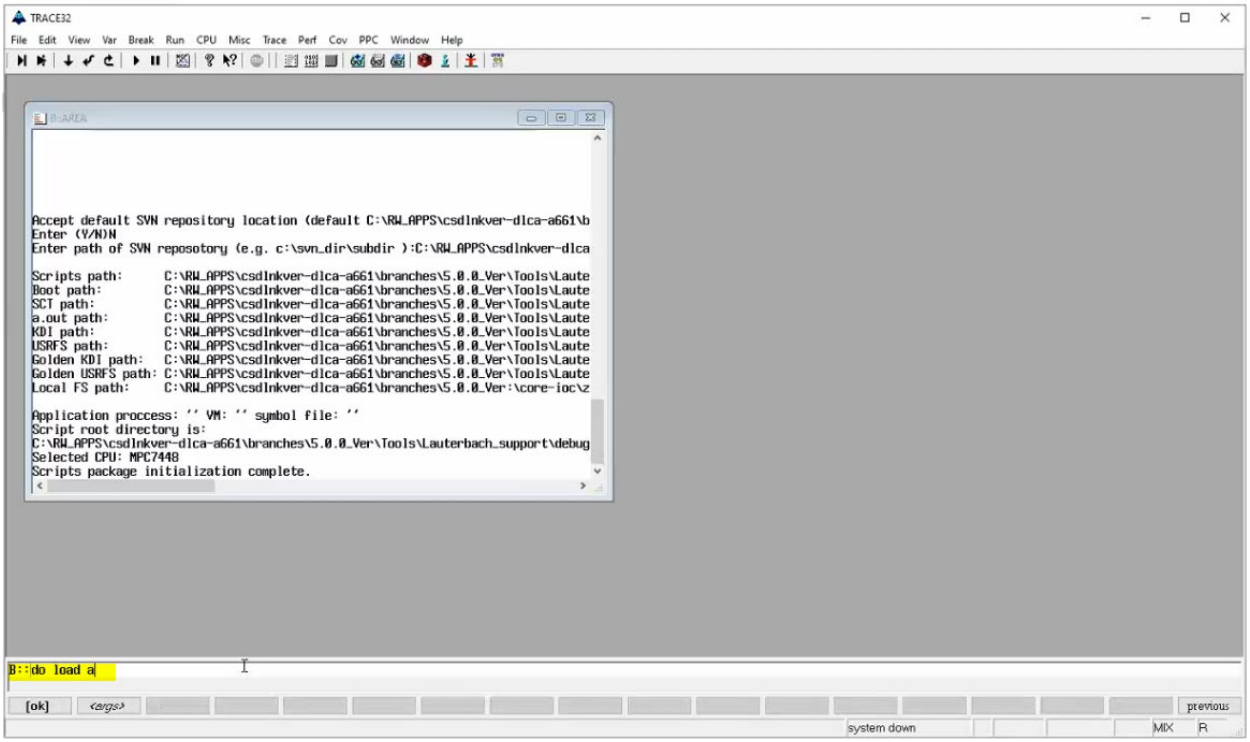




12 Confirm the proper files are selected. There are a couple of things to note on this screen. Ensure that kdi, a.out and usrfs are pointing to dev versions, as prod versions will not work with BASH prompt loads. Check 'local FS' is pointing to the correct location for the load you are doing. Currently there are unique locations for different project's which loads. Select OK.



13 Turn off watchdog on the CPA_Disc screen (WD_INH_F)
14 Execute 'do load a' from the command prompt.



15 When the screen looks like the following the, CCM is successfully loaded.

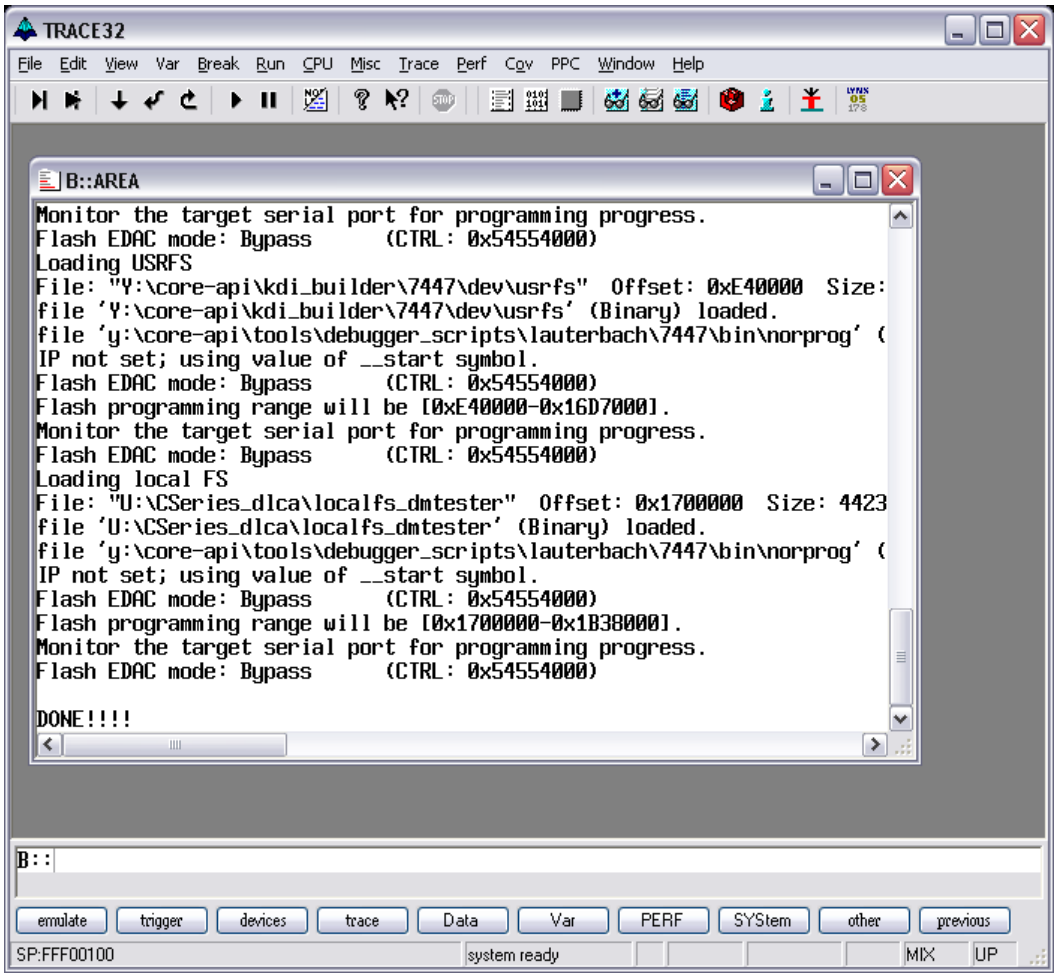
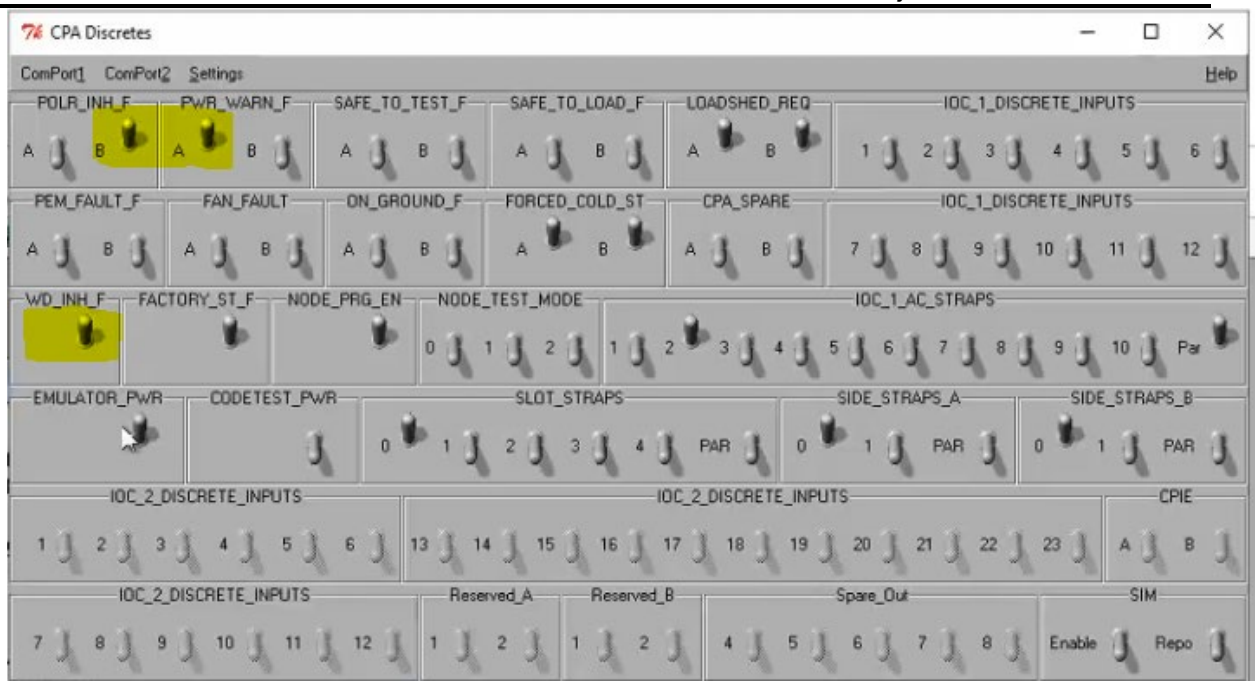


Figure 50: TRACE32 after CCM is successfully loaded

16. Open CPA discretes, Turn watchdog (WD_INH_F) back on.
17. Power off the discretes PWR_WARN_F switch A and POLR_INH_F switch B.



18. Open COM6 and COM7 teraterm windows.
19. Power On the discretes PWR_WARN_F switch A and POLR_INH_F switch B
20. Enter below commands without quotes, by right clicking on bash command prompt in teraterm console COM6, at '[singleuser: /] $'
    "sw_val_complete"
    "cinit cnc"
    Note: Bash prompt will come for 3 times in teraterm console, if the commands are given properly With in these attempts, then steps 17 to 20 need to be followed untill successfully completed
21. Enter 'pm' in the TeraTerm console to start Protocol manager.
    In COM7 Tera Term window,
22. Enter 'dlca' in the other console to start DLCA. The startup process will initialize all of the data flows and return 'Init done' when successfully completed.Make sure that the partnumbers are correct or not.

8.5.3.2 Right DLCA

1. To perform a Lauterbach load, a target build will need to have been successfully completed as per the guidance in the section 'Custom Builds Instruction' .
2. Open two TeraTerm terminals, one for each of the VM's on the load. Typically PM and HM will output to COM4 and DLCA will output to COM2.
3. Open CPA_Disc.exe, as shown and ensure that the ComPort is select as COM5.

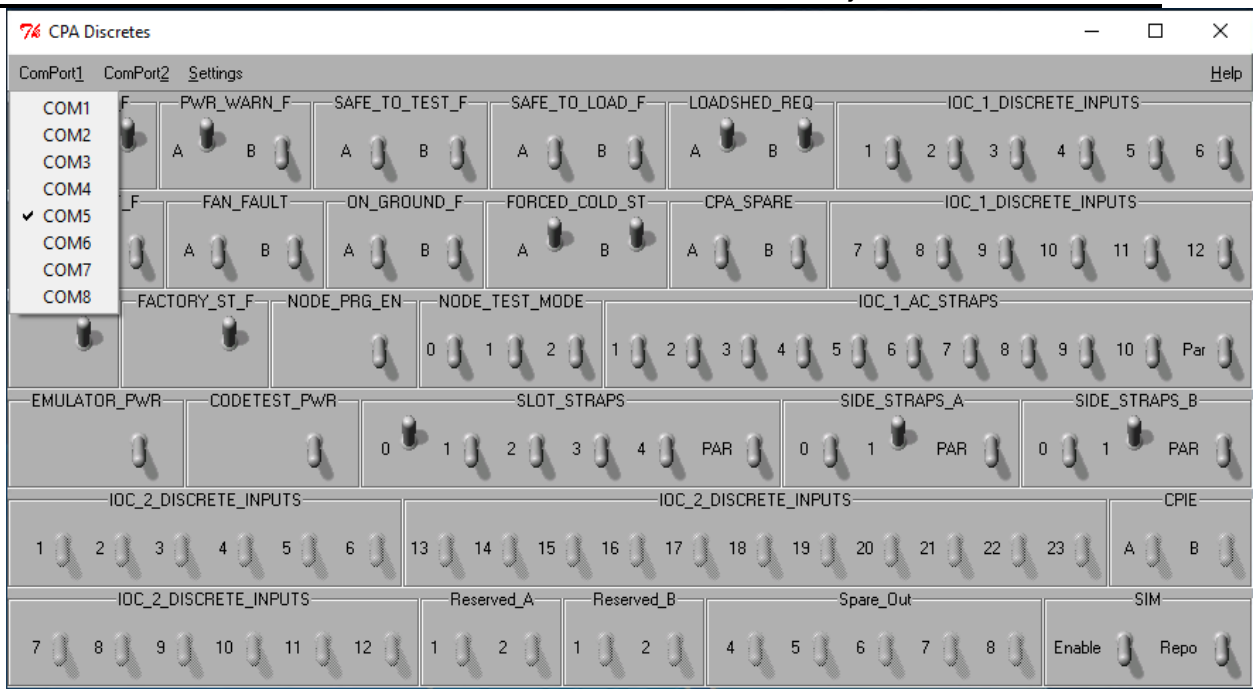
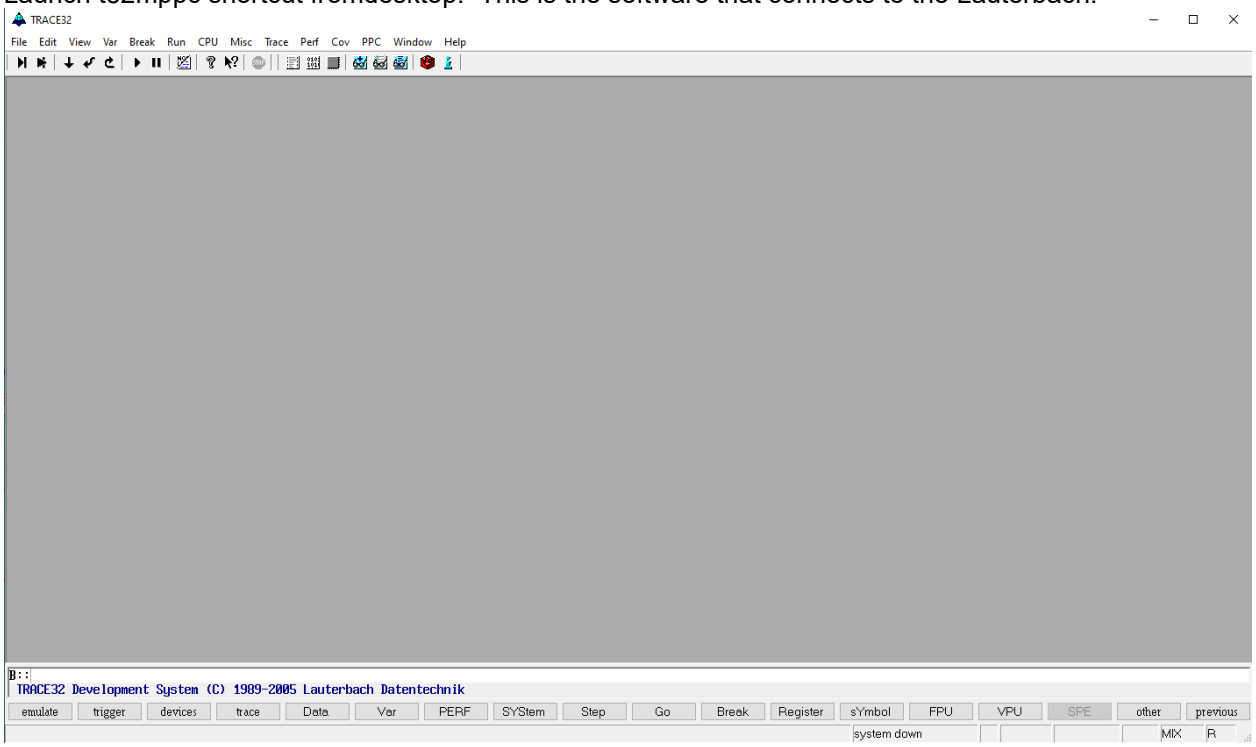
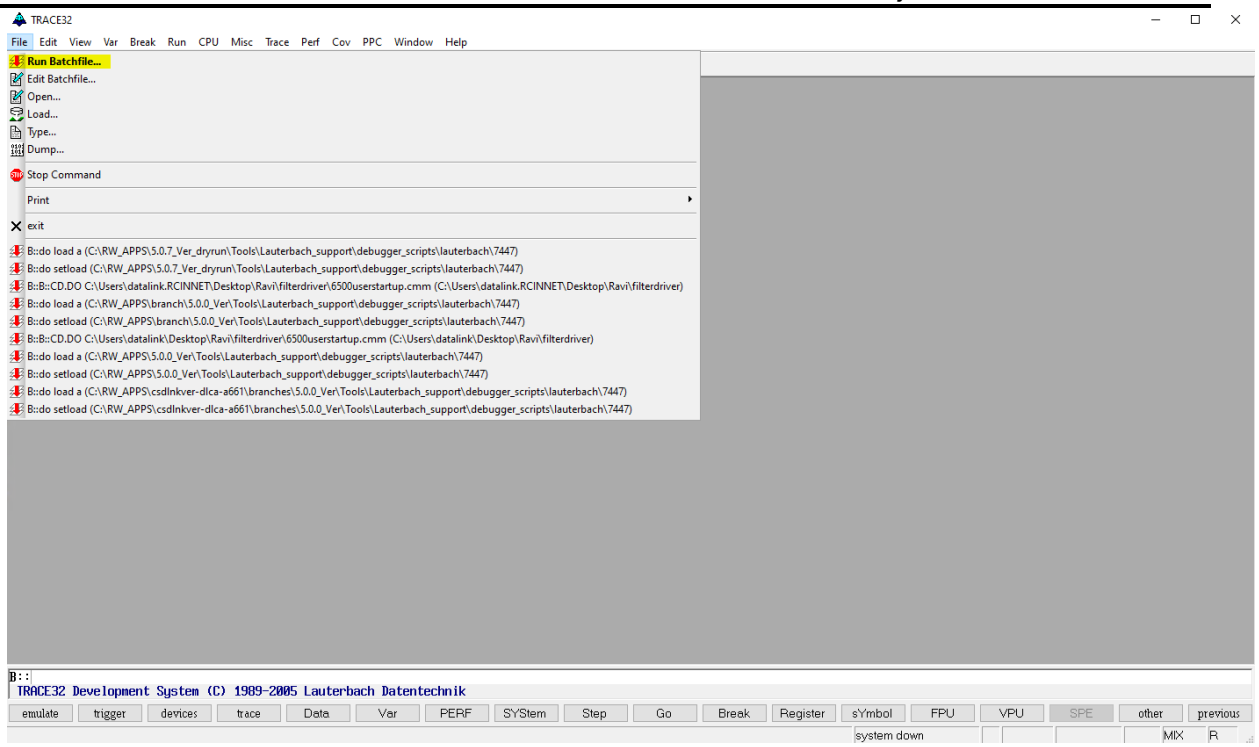


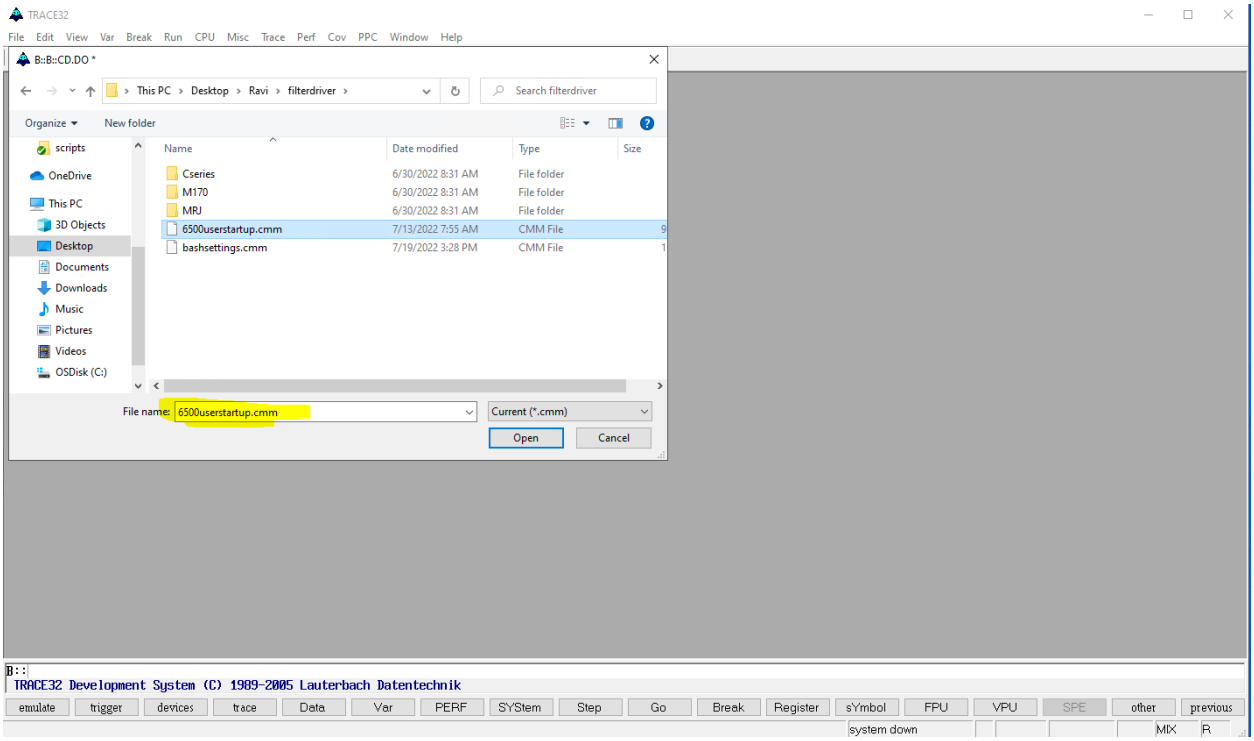
Figure 51: CPA Discretes

4. Ensure that Turn watchdog (WD_INH_F) is on.
5. AFDX Filter Driver : Follow the steps as per the section " Configuring AFDX Network"
6. Launch t32mppc shortcut fromdesktop. This is the software that connects to the Lauterbach.

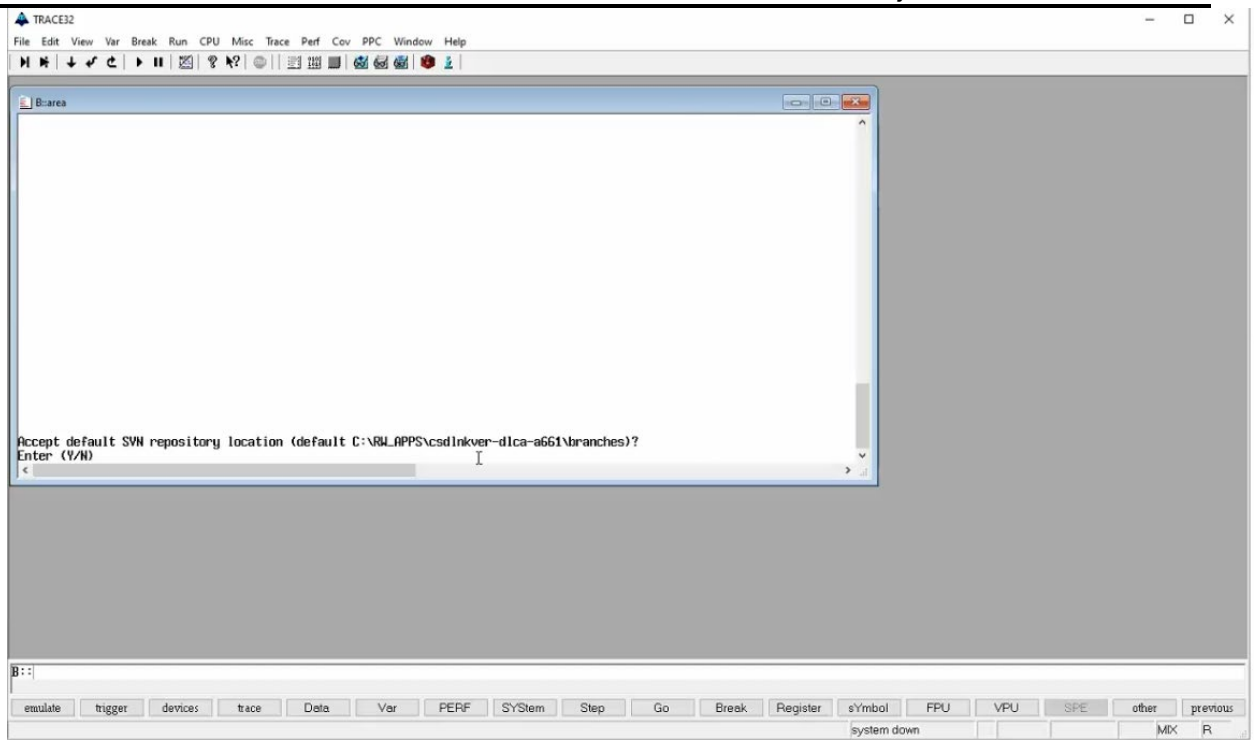


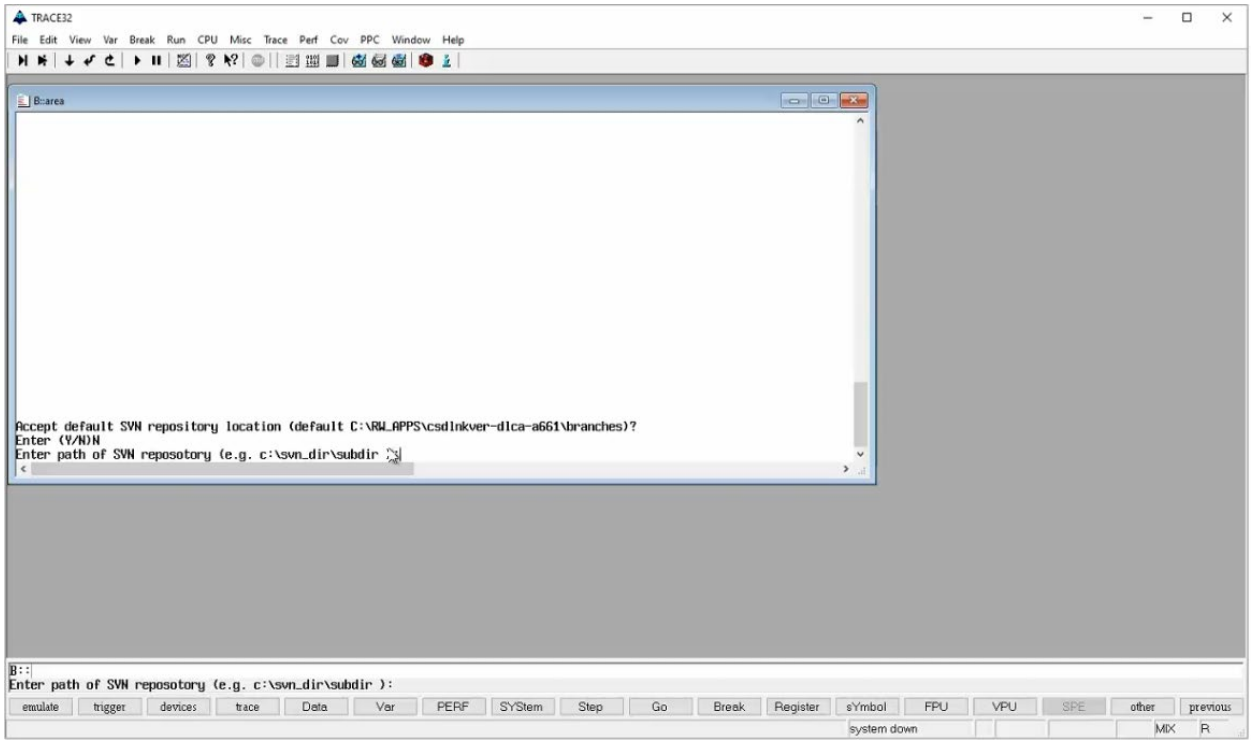
7. Run the "6500Userstartup.cmm" file from File->Run Batchfile

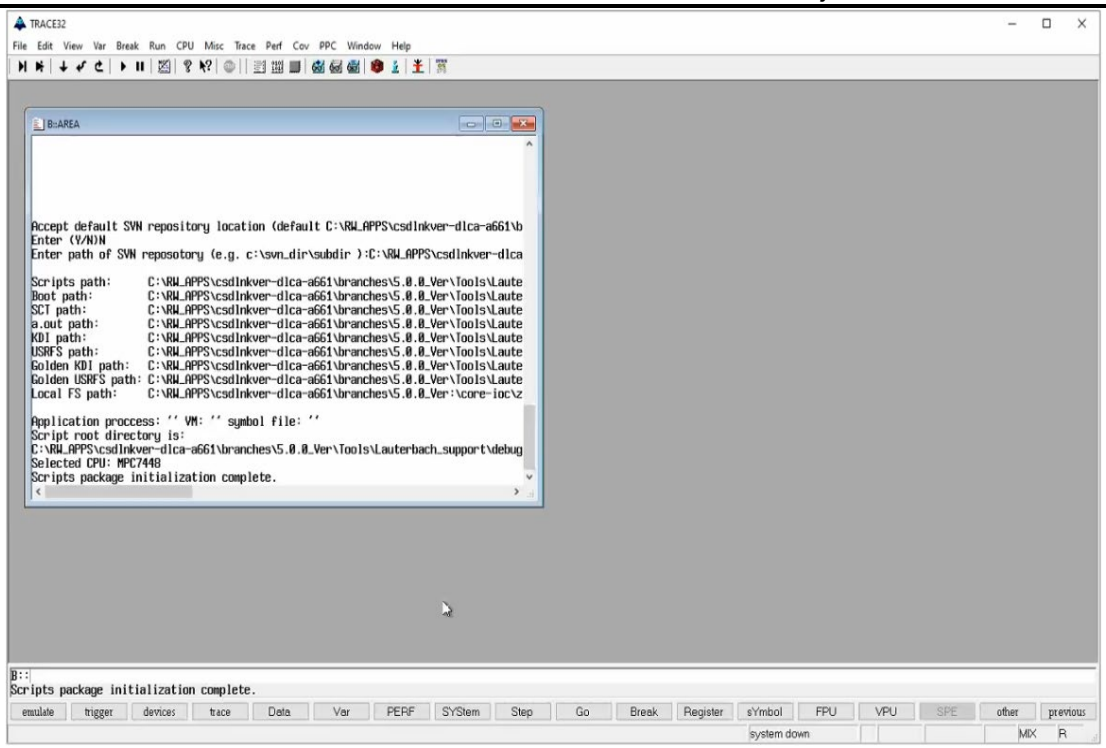




8. Enter 'Y' if the default SVN repository location shown is correct or Enter 'N' and Enter path of SVN repository as https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/







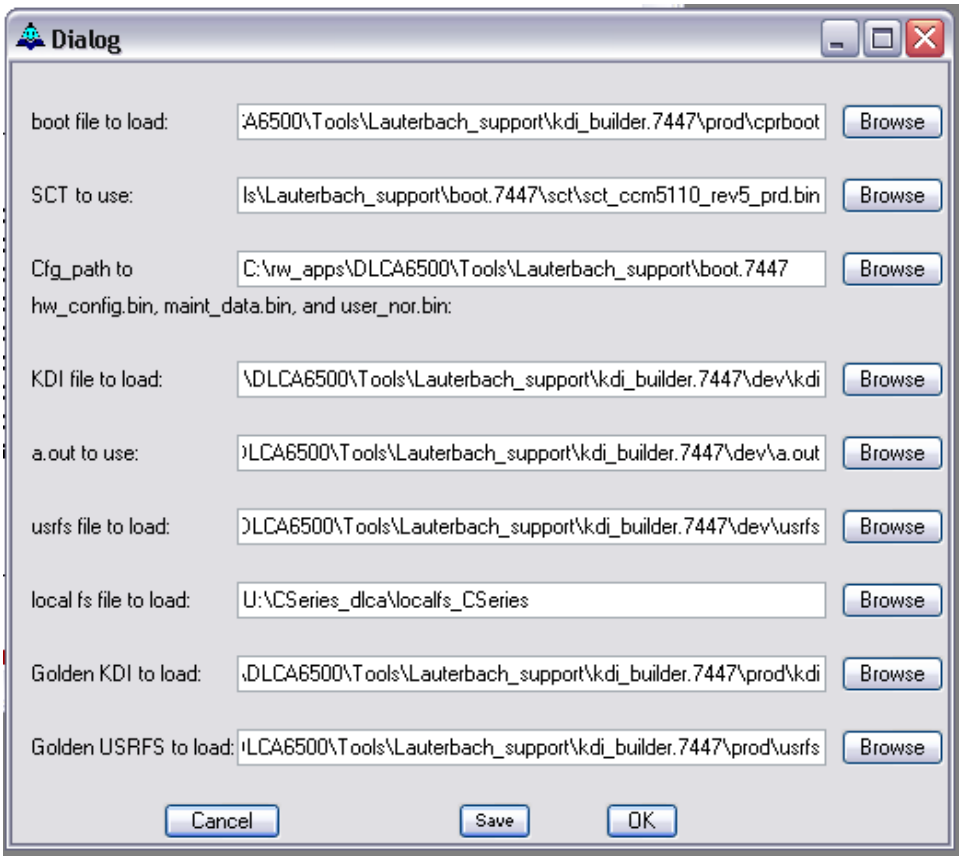
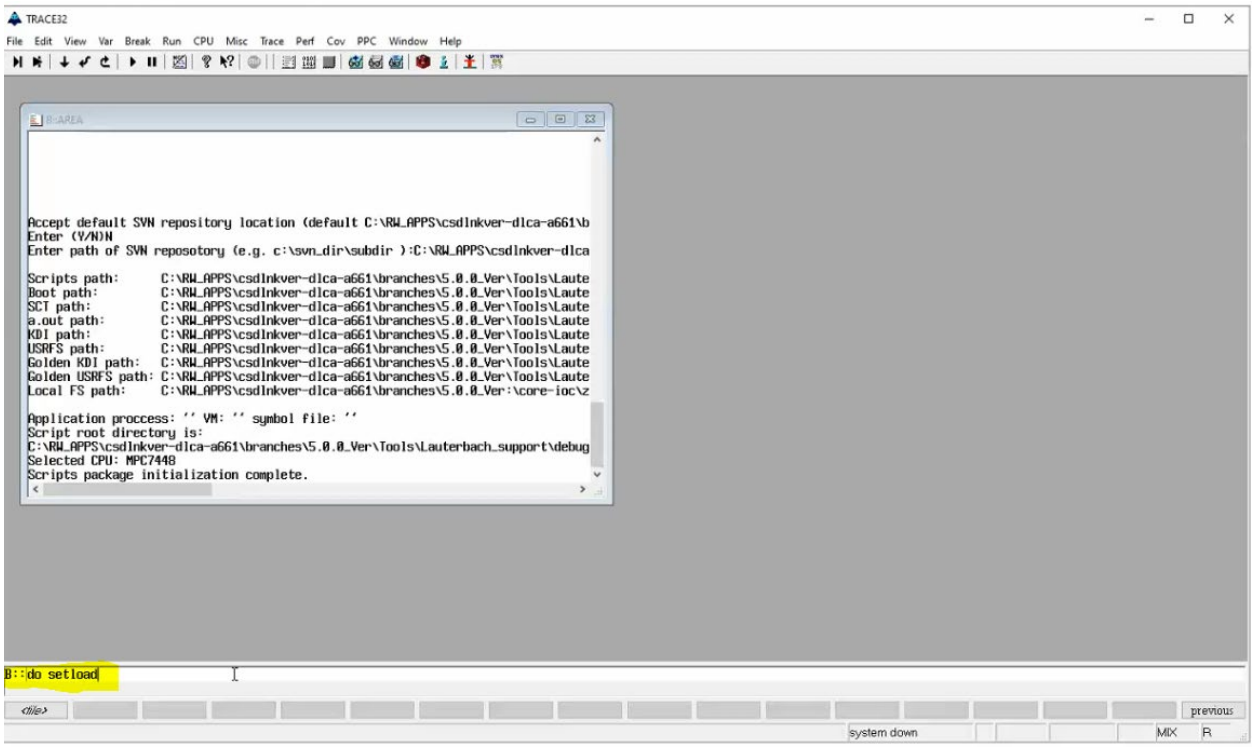
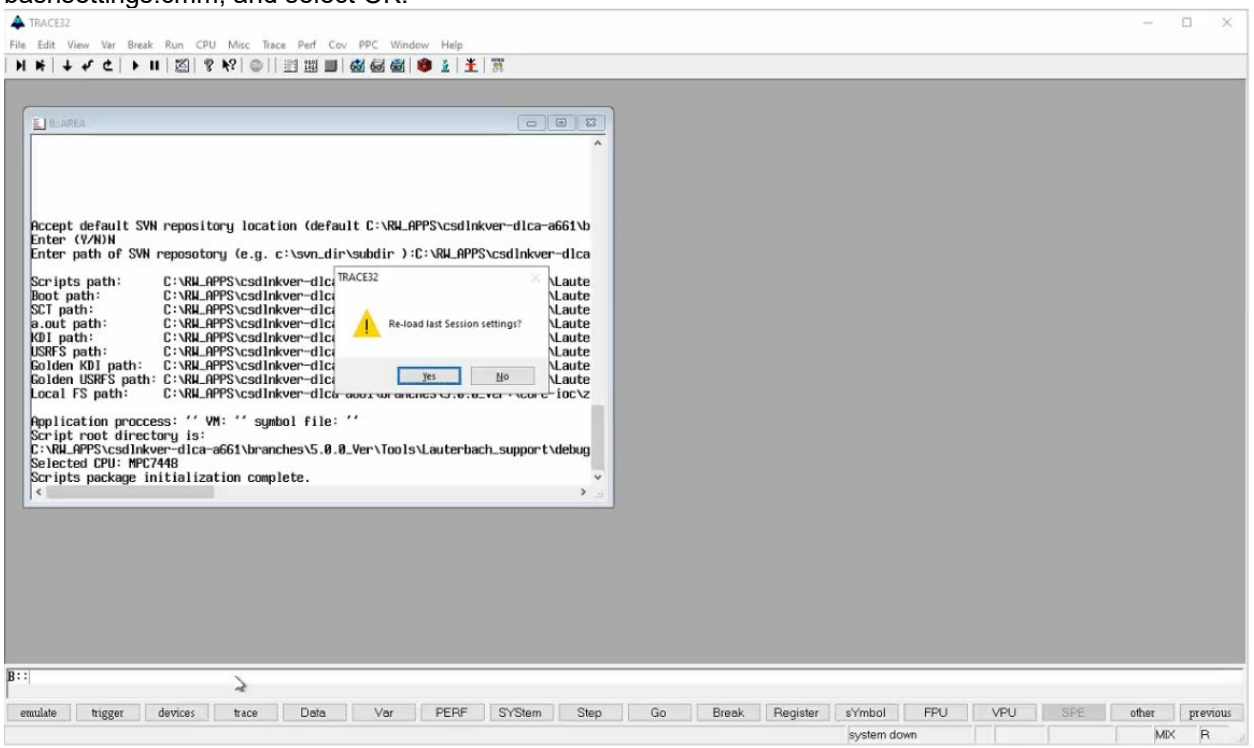


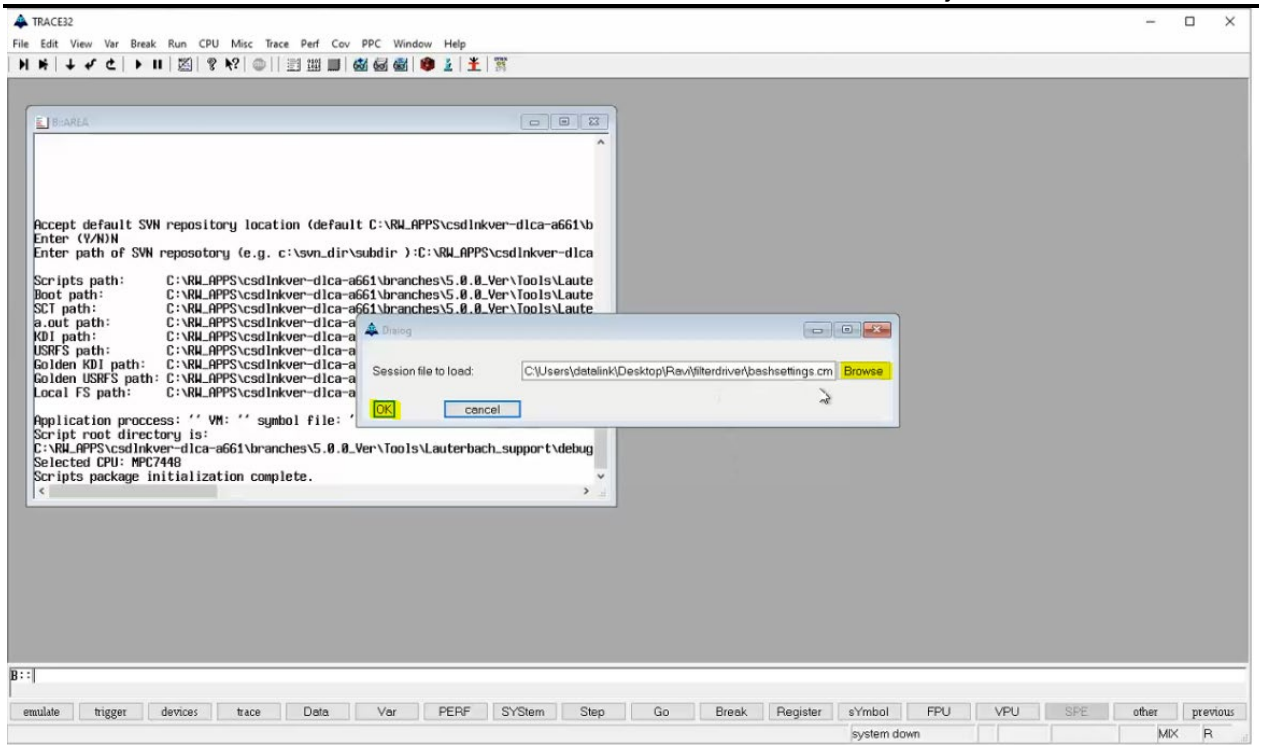
Figure 52: Dialog box DLCA6500 load

9. After seeing message, 'Scripts package initialization complete', Execute 'do setload' from the Trace32 command prompt.

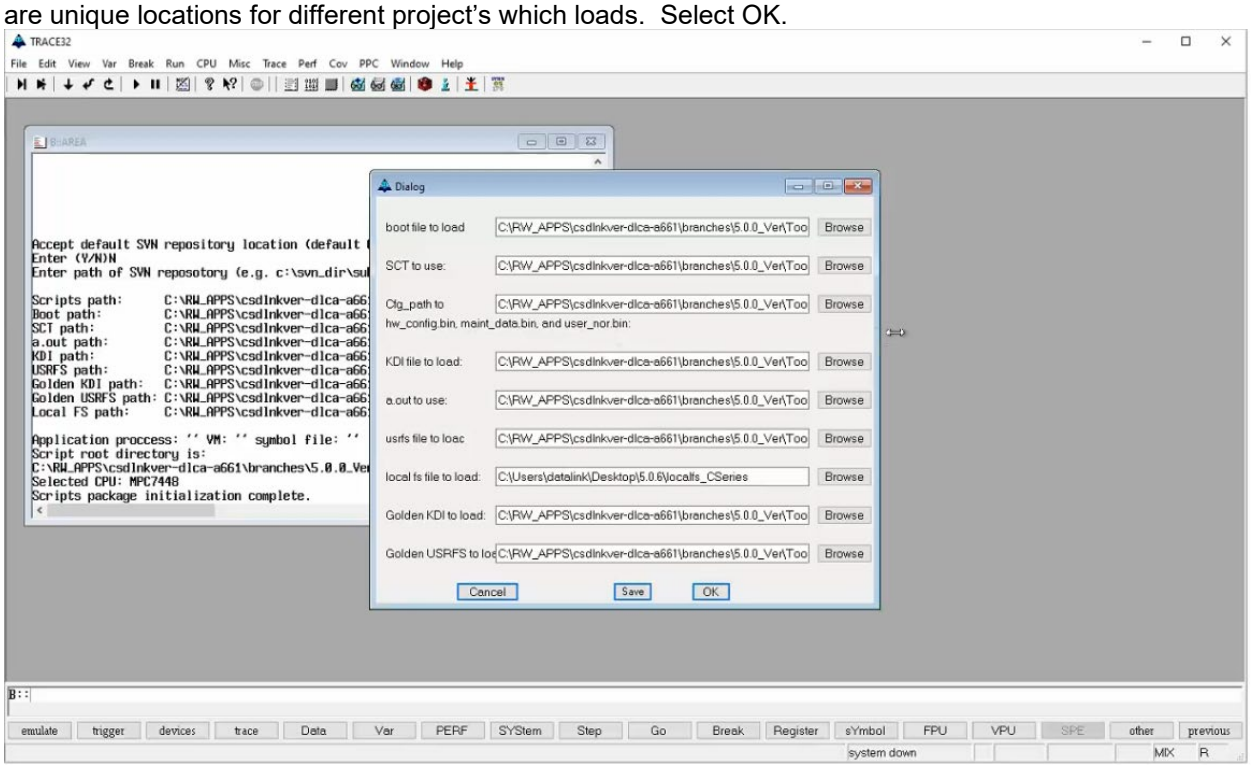


10. Choose 'Yes' to the Reload last Session settings popup, and browse to the setttings file bashsettings.cmm, and select OK.

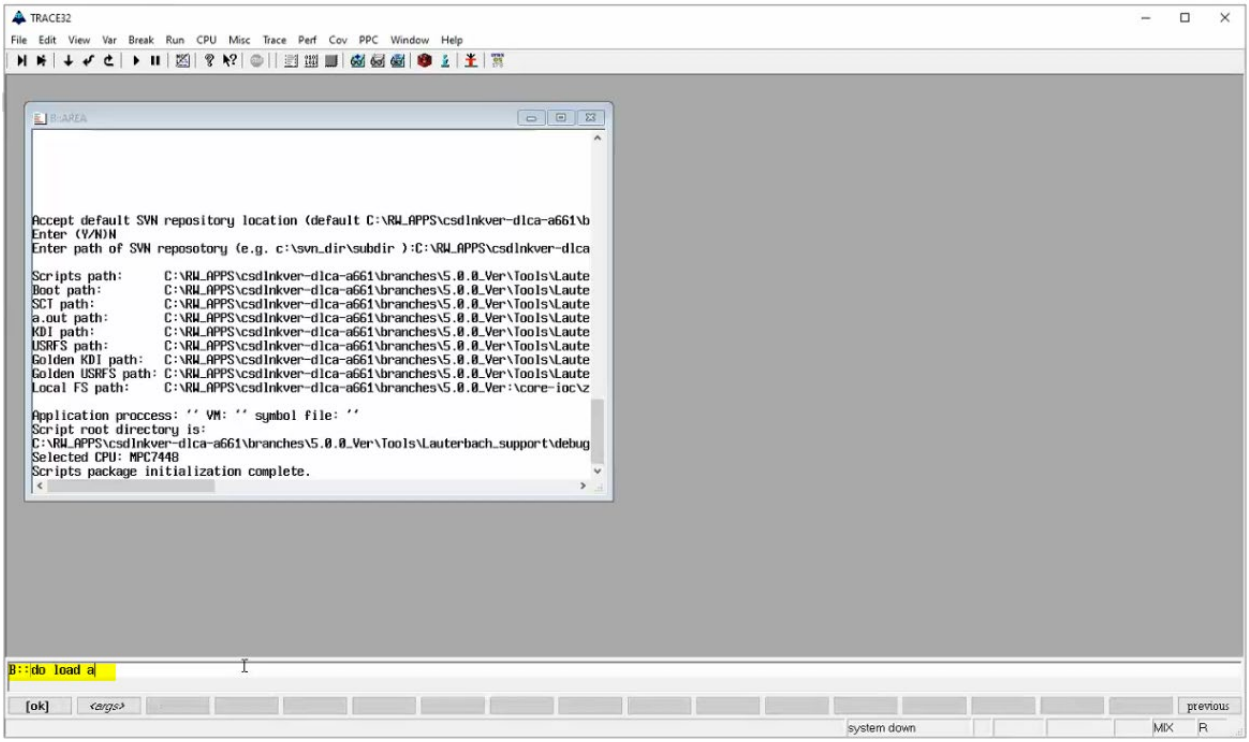




11. Confirm the proper files are selected. There are a couple of things to note on this screen. Ensure that kdi, a.out and usrfs are pointing to dev versions, as prod versions will not work with BASH prompt loads. Check 'local FS' is pointing to the correct location for the load you are doing. Currently there are unique locations for different project's which loads. Select OK.



12. Turn off watchdog on the CPA_Disc screen (WD_INH_F).
13. Execute 'do load a' from the command prompt.



14. When the screen looks like the following the, CCM is successfully loaded.

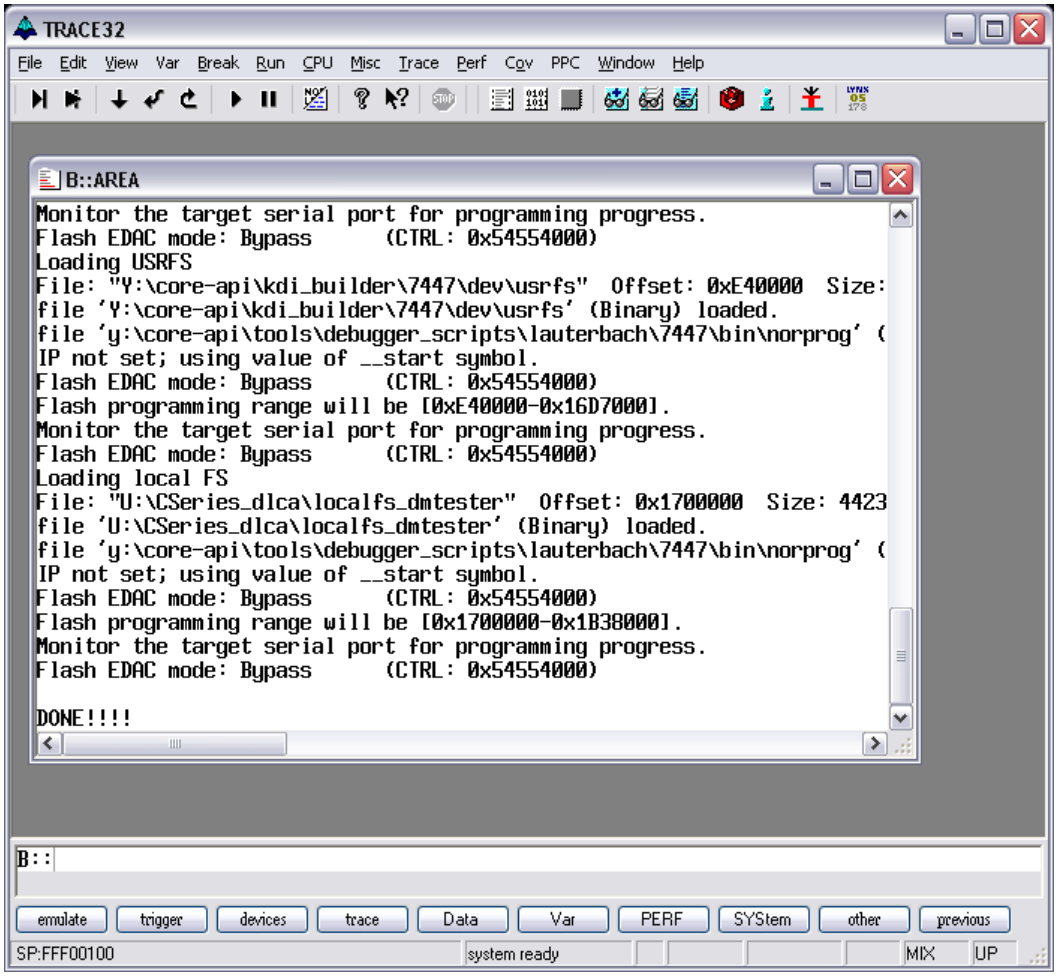
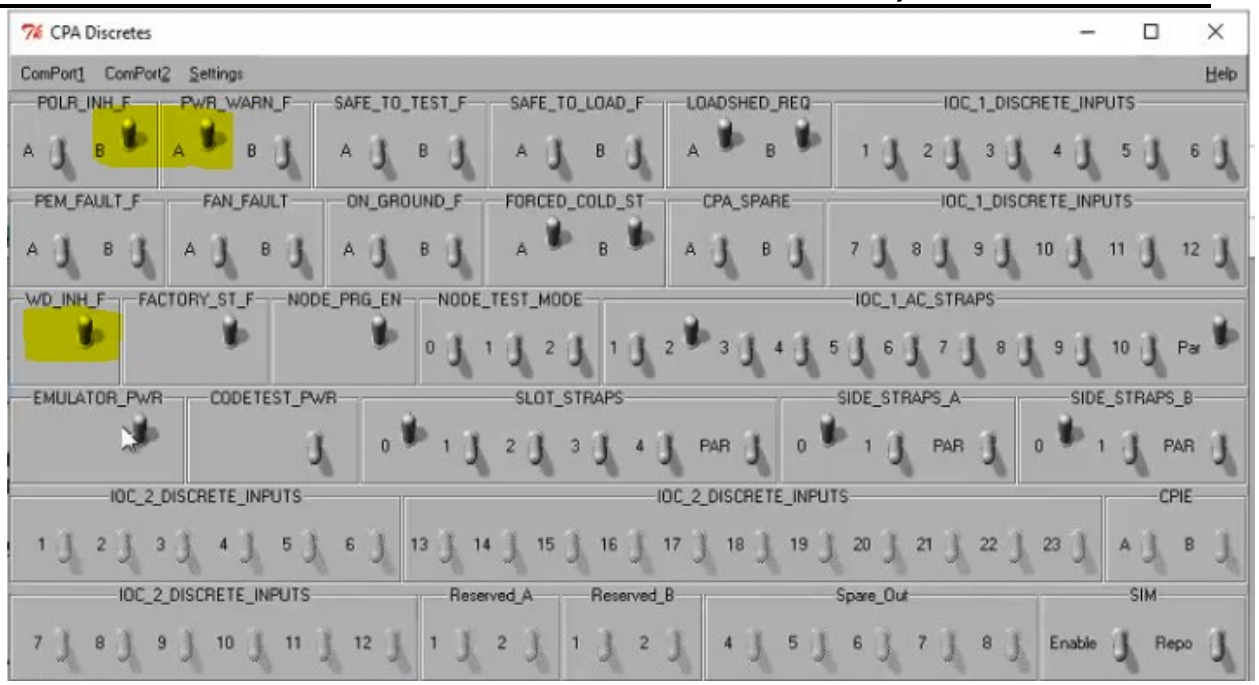


Figure 53: TRACE32 after CCM is successfully loaded

15. Open CPA discretes, Turn watchdog (WD_INH_F) back on.
16. Power off the discretes PWR_WARN_F switch A and POLR_INH_F switch B.



17. Open COM4 and COM2 teraterm windows.
18. Power On the discretes PWR_WARN_F switch A and POLR_INH_F switch B
19. Enter below commands without quotes, by right clicking on bash command prompt in teraterm console COM4, at '[singleuser: /] $'
    "sw_val_complete"
    "cinit cnc"
    Note: Bash prompt will come for 3 times in teraterm console, if the commands are given properly With in these attempts, then steps 16 to 19 need to be followed untill successfully completed
20. Enter 'pm' in the TeraTerm console to start Protocol manager.
21. In COM2 Tera Term window, Enter 'dlca' to start DLCA. The startup process will initialize all of the data flows and return 'Init done' when successfully completed.Make sure that the partnumbers are correct or not.
22. After this, Close all tera term and cpa discrete windows.

8.5.3.4 Starting Target

1. Start the VISTA environment in the https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/vista_sim_IPSDual/System_target.bat directory of SVN to start all of the hosted applications that connect to DLCA and PM.
2. Check for DLCA and PM Health Status. If it is off , Terminate the Vista environment and re-run A_PM_create.ttl, B_PM_create.ttl, L_DLCA_create.ttl and R_DLCA_create.ttl files
   [Refer to section 8.5.2.5 TTL files for Target, for svn path of the files]

8.6 Target-based Test Station(EDS- Dual DLCA)

8.6.1 Folder/exe Locations

TAM Discrete - <Verification- Branch>/tools/EDS_Env_Support/tam_discretes

Tera Term - <Verification-Branch>/tools/Target_Tools/TeraTerm

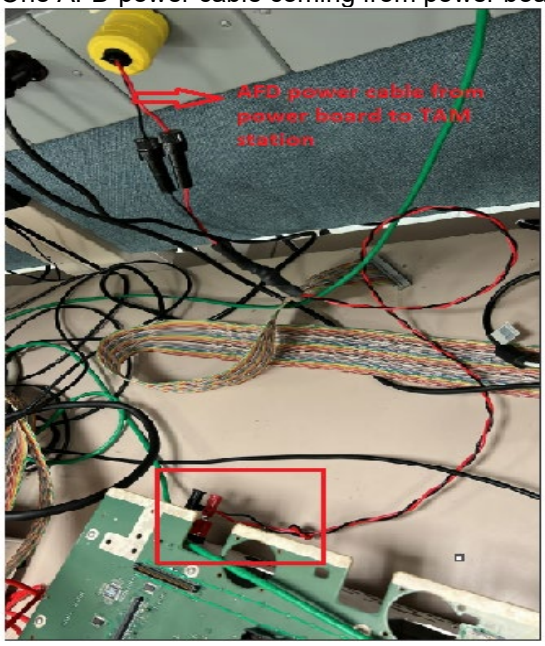
8.6.2 Configuration Details

8.6.2.1 Physical Connection Details

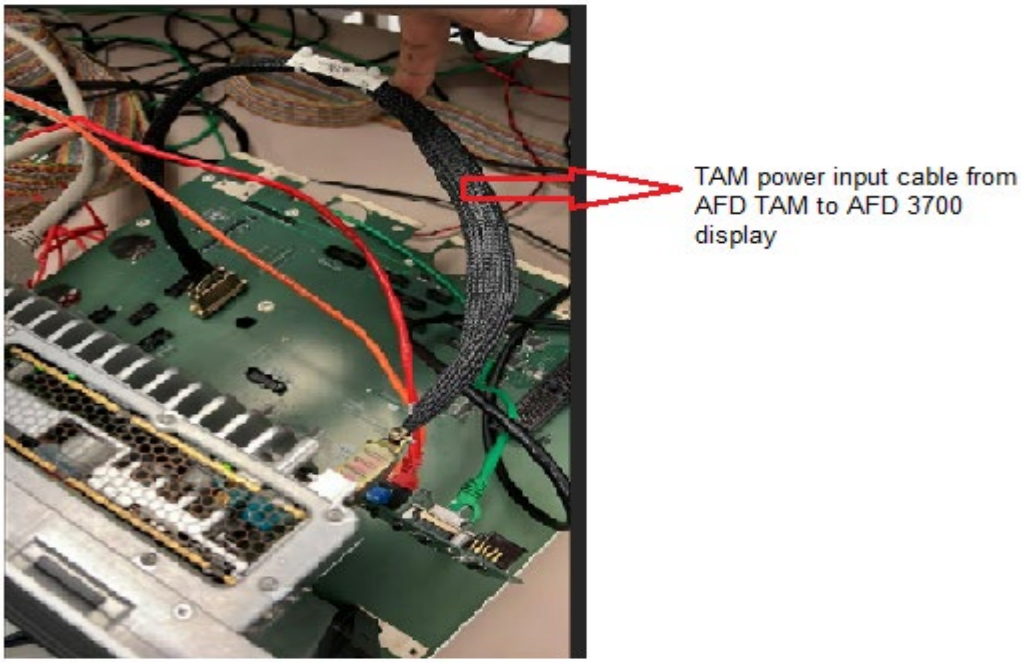
To begin with, the following cables and equipment is required to set up the dual target system.

8.6.2.1.1 Left AFD TAM Station:

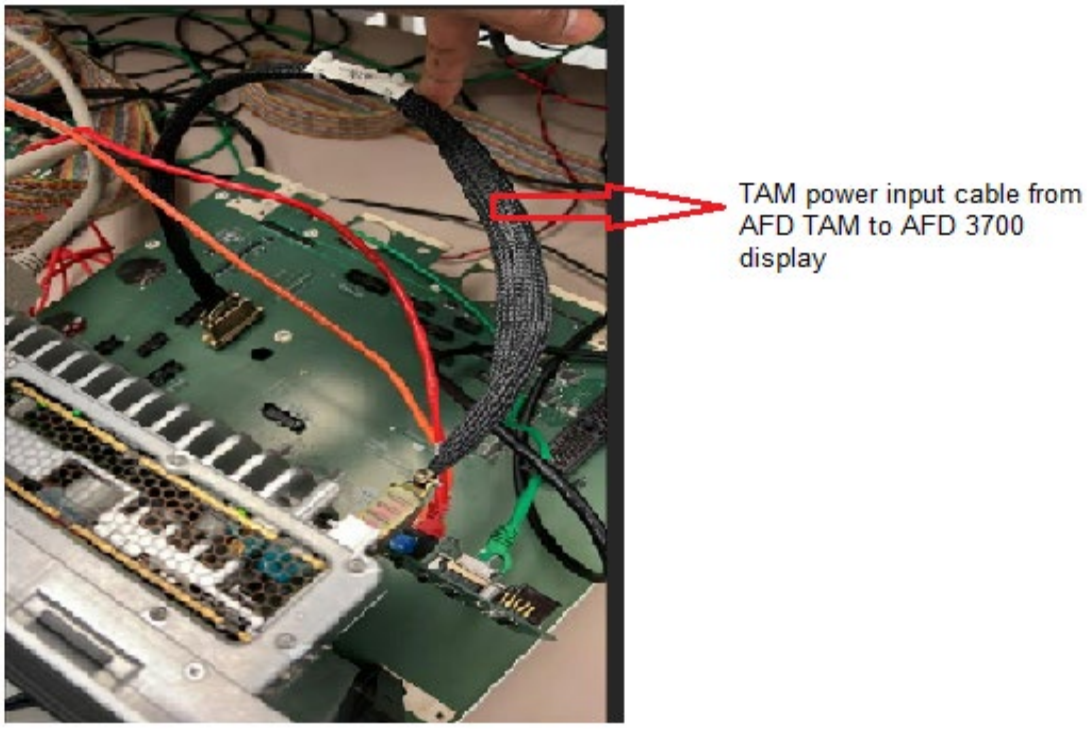
- One AFD power cable coming from power board and going to the Left AFD TAM



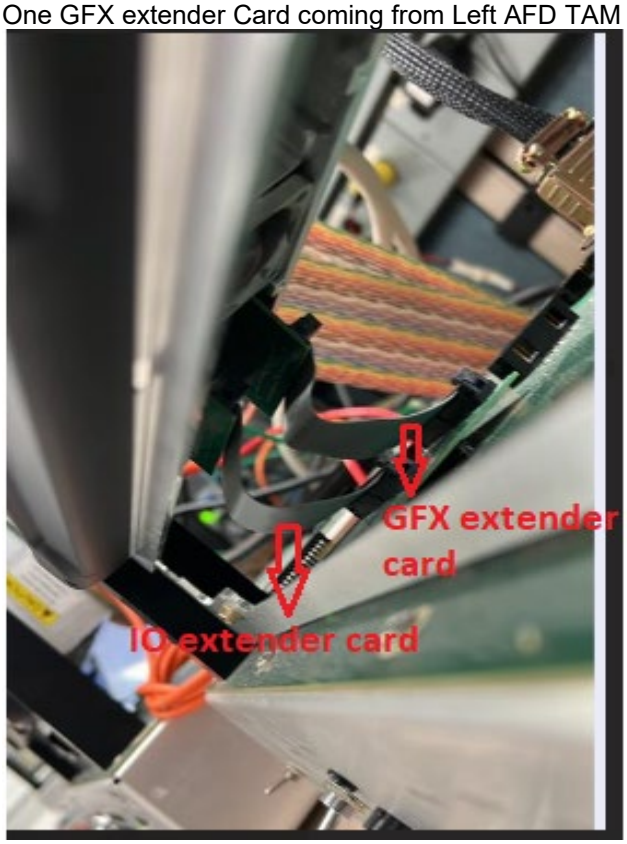
- One TAM power input cables coming from Left AFD TAM and going to the Left AFD 3700 display



- One TAM extender Card coming from Left AFD TAM and going to the Left AFD 3700 display



- One IO extender Card coming from Left AFD TAM and going to the Left AFD 3700 display
- One GFX extender Card coming from Left AFD TAM and going to the Left AFD 3700 display



- One Dataload cable and one TestInterface cable coming from Left AFD J3 Wraparound board(COM5 and COM 7 respectively) and going to the VLAN ports of ethernet hub/switch. Assign the IP address and subnet mask value as per below,

Dataload(COM5):

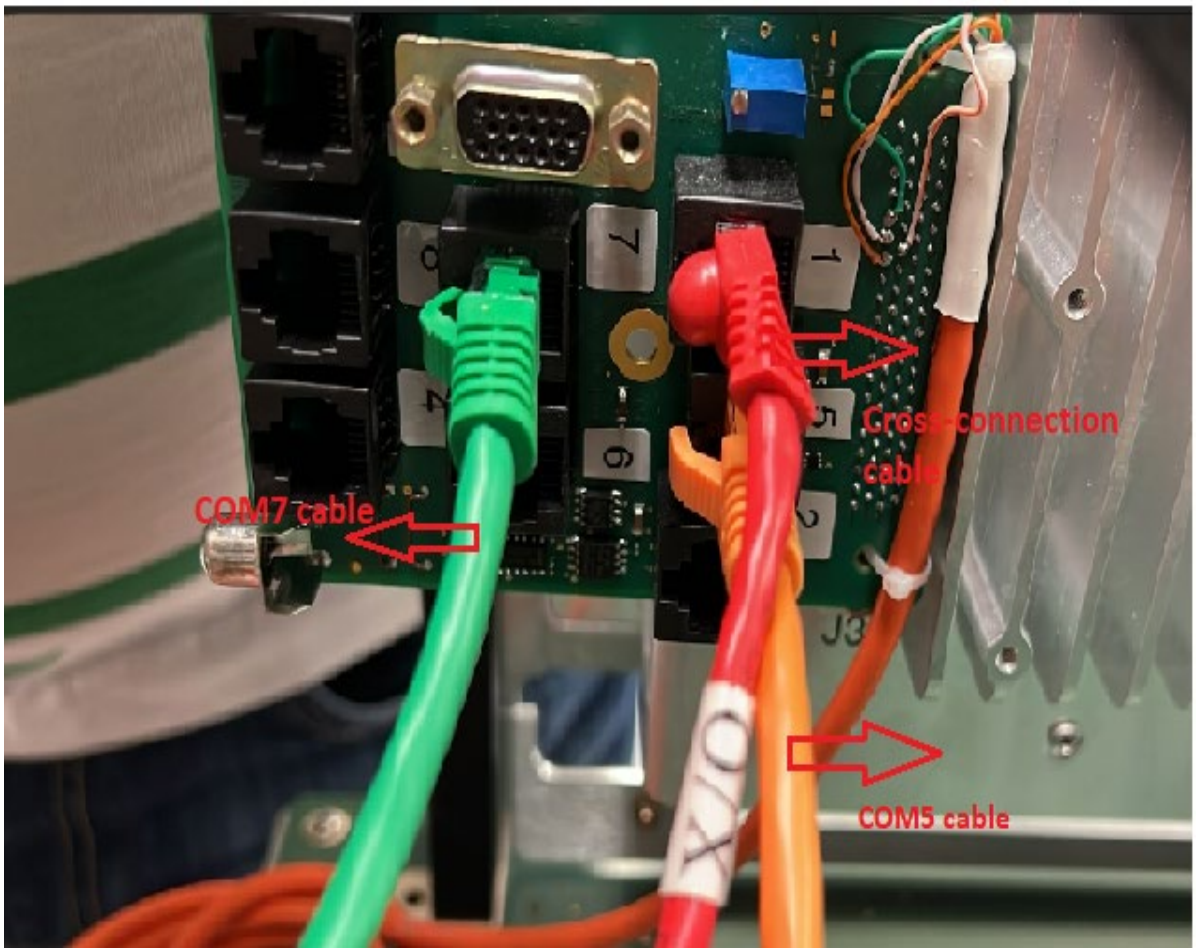
IP address: 10.129.25.0

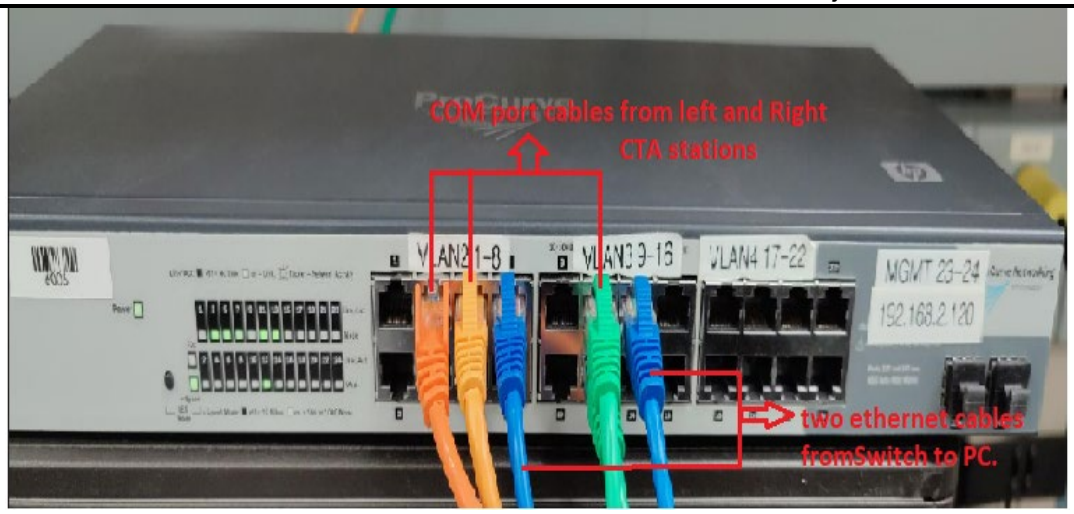
Subnet mask: 255.224.0.0

TestInterface(COM7):

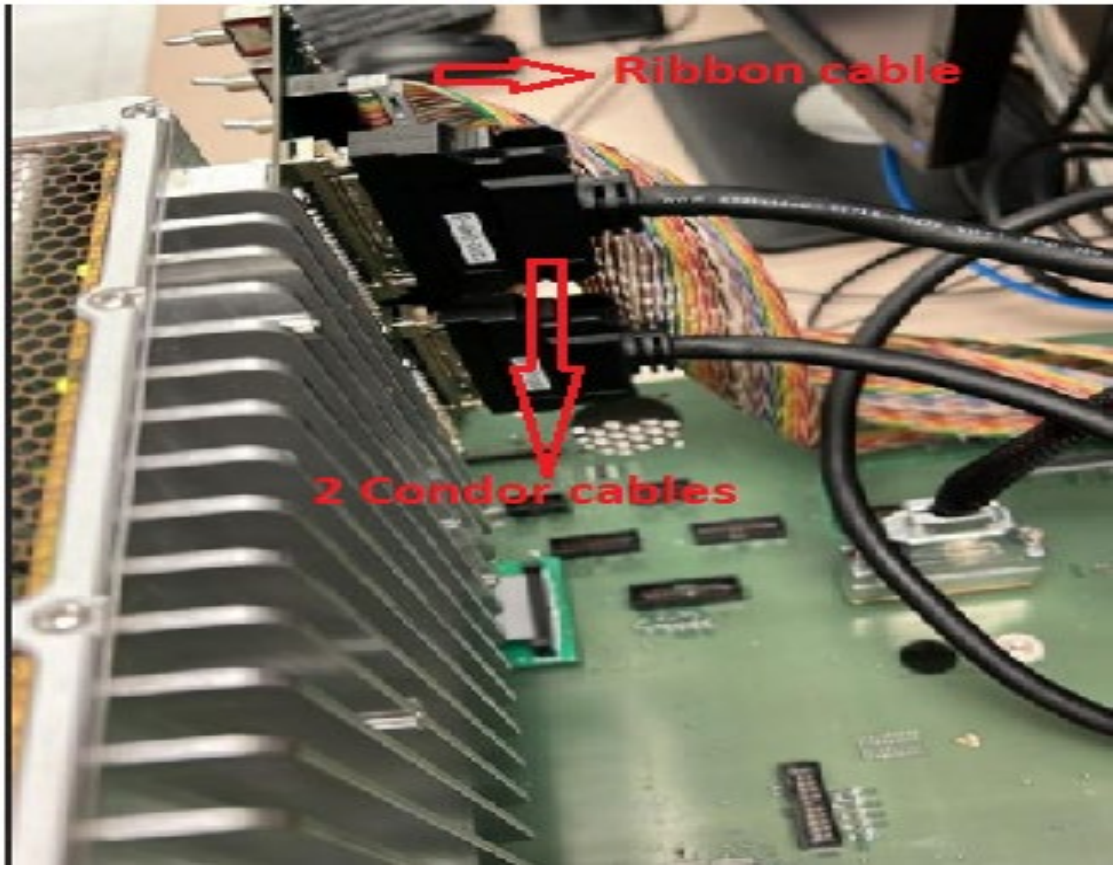
IP address: 10.145.26.1

Subnet mask: 255.255.255.0

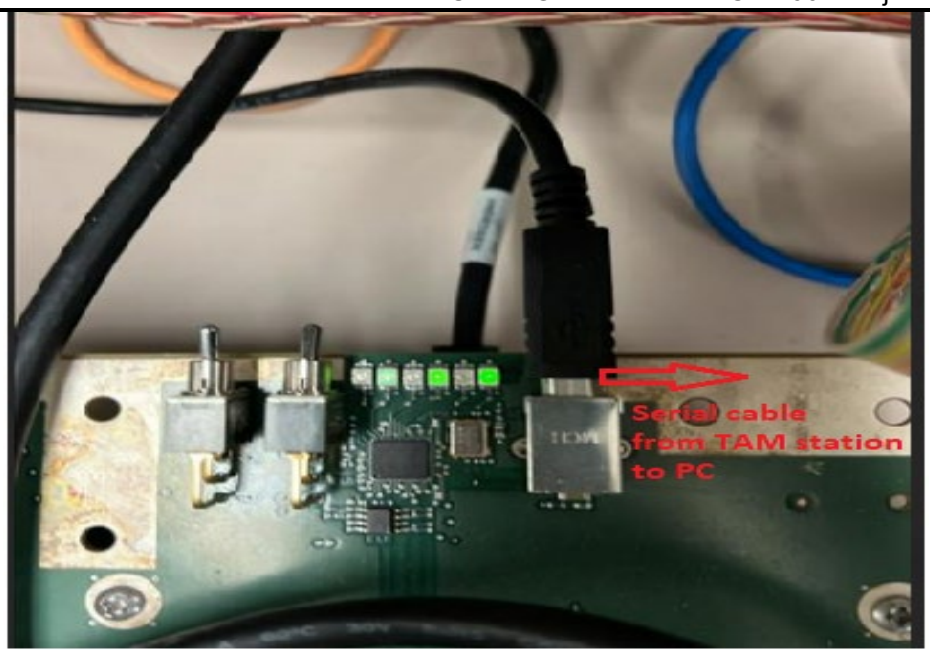




- Two Condor cables coming from the Left AFD TAM Discrete card and going to the serial ports on the PC (two per PC)
- One Ribbon cable coming from the Left AFD TAM Discrete card and going to the PC



- One serial from the TAM station to the target PC.



- One Ethernet cable coming from NIC on the PC and going to the VLAN port of ethernet hub/switch

Note: Images mentioned under Left AFD TAM station are similar to Right TAM station.

8.6.2.1.2 Right AFD TAM Station:

- One AFD power cable coming from power board and going to the Right AFD TAM
- One TAM power input cables coming from Right AFD TAM and going to the Right AFD 3700 display
- One TAM extender Card coming from Right AFD TAM and going to the Right AFD 3700 display
- One IO extender Card coming from Right AFD TAM and going to the Right AFD 3700 display
- One GFX extender Card coming from Right AFD TAM and going to the Right AFD 3700 display
- One Dataload cable and 1 TestInterface cable coming from Right AFD J3 Wraparound board(COM5 and COM 7 respectively) and going to the VLAN ports of ethernet hub/switch. Assign the IP address and subnet mask value as per below,

  Dataload(COM5):

  IP address: 10.129.25.0

  Subnet mask: 255.224.0.0

  TestInterface(COM7):

  IP address: 10.145.26.1

  Subnet mask: 255.255.255.0

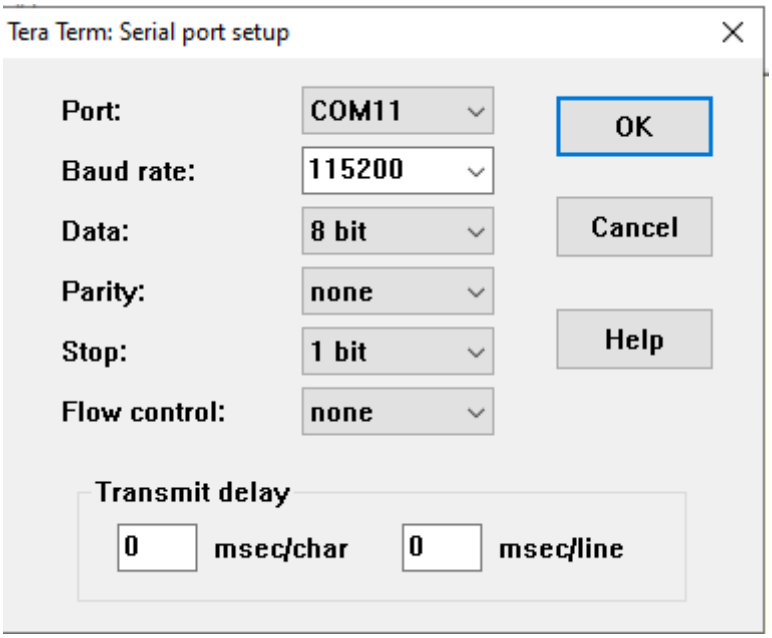
- Two Condor cables coming from the Right AFD TAM Discrete card and going to the serial ports on the PC (two per PC)
- One Ribbon cable coming from the Right AFD TAM Discrete card and going to the PC
- One Ethernet cable coming from NIC on the PC and going to the VLAN port of ethernet hub/switch

8.6.2.1.3 Cross Talk connection between Left AFD TAM and Right AFD TAM stations:

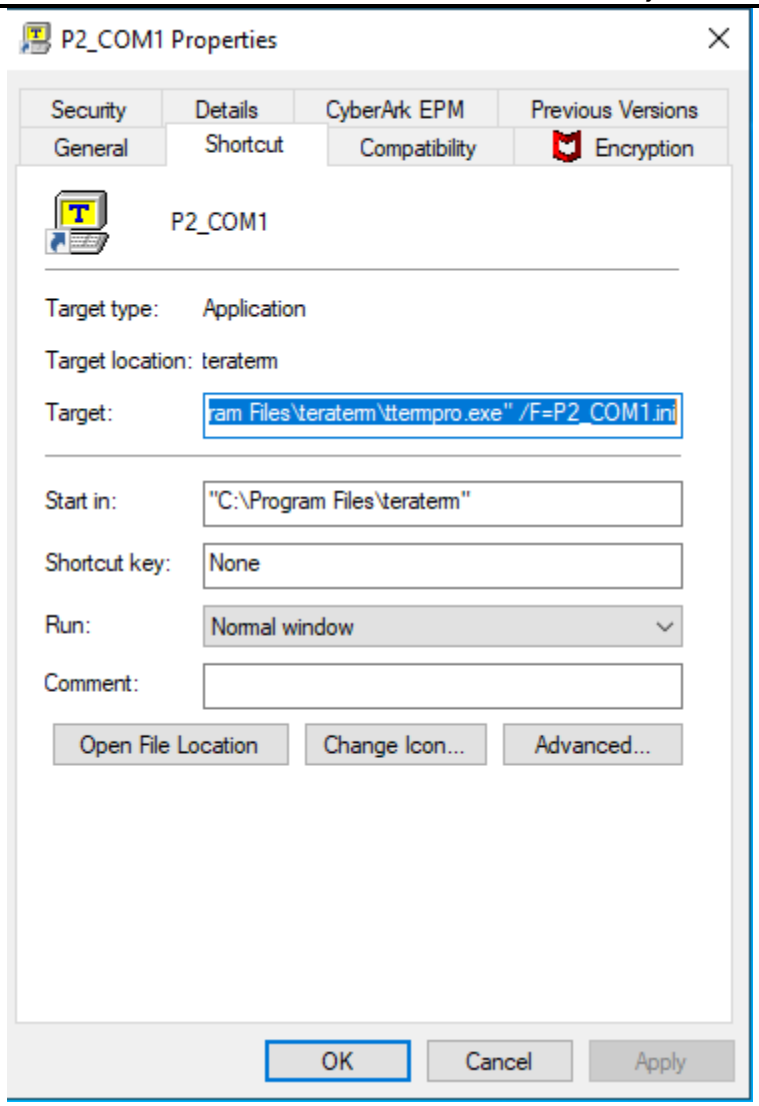
- One Crosstalk cable coming from Left AFD J3 Wraparound board (COM1) and going to the Right AFD J3 Wraparound board (COM1)

8.6.2.2 Configuring the Tera Term files

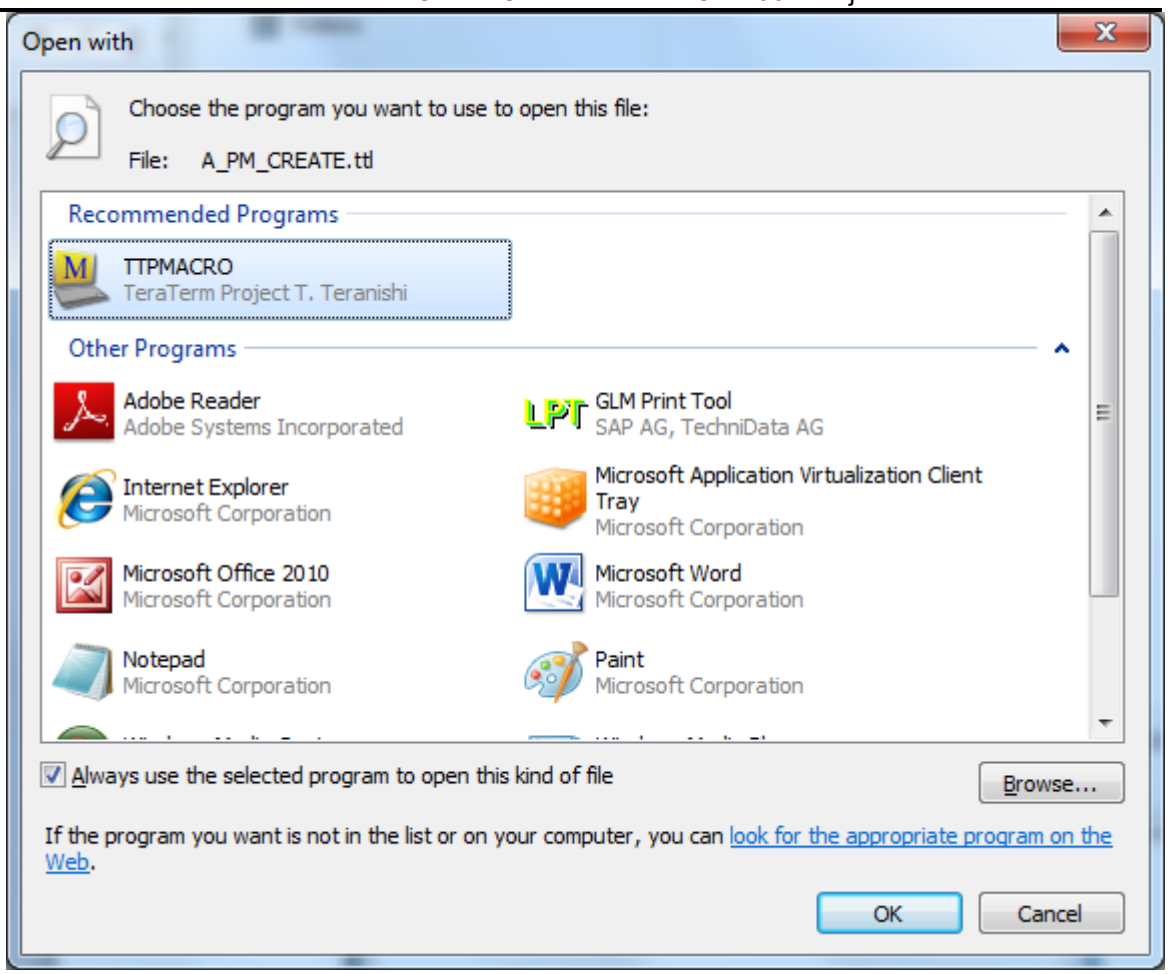
1. Copy the teraterm folder to C:\Program Files (x86)
2. Create 2 desktop short cuts to C:\Program Files (x86)\teraterm\ttermpro
3. Open the Tera Term short cut, in setup change the serial port settings for COM11(Left PM) as shown below



4. Similarly change the serial port settings for COM5(Left DLCA).
5. In Desktop rename the shortcuts as 'L_P5_COM1' for Left PM and 'L_P2_COM1' for Left DLCA.
6. Right click on the L_P2_COM1 short cut and select properties
7. Set the INI file in target path as shown below



8. Similarly Set the INI file in target for L_P5_COM1.
9. Repeat the steps from 3 to 8 for COM23 and COM17
10. For Verification, open folder https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/vista_sim_EDSDual/scripts
11. Open A_PM_CREATE.ttl file. Browse to path C:\Program Files (x86)\teraterm\ttpmacro and select check box "always open the program"

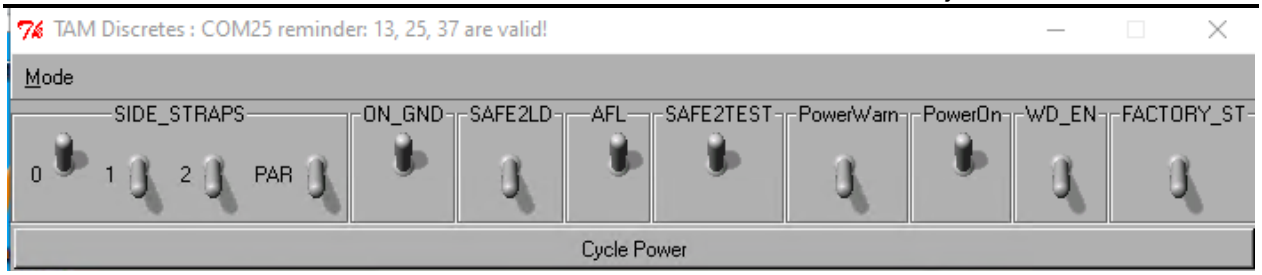


COM PORT NAME	APPLICATION
COM_11	Left PM
COM_5	Left DLCA
COM_23	Right PM
COM_17	Right DLCA

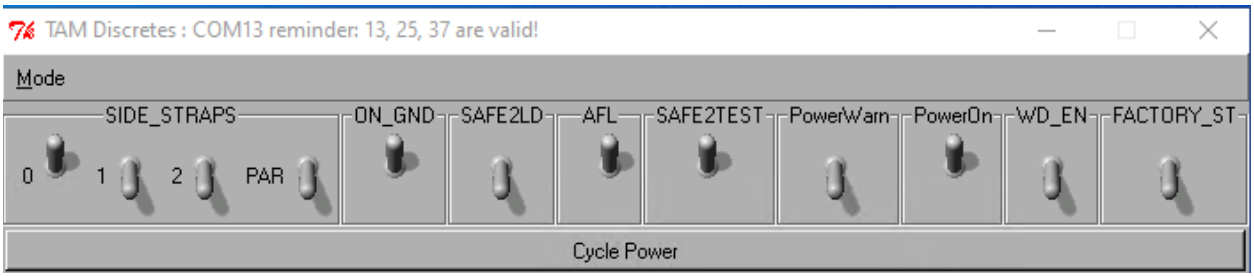
Note: The COM ports are configured locally on PC CRP27090. COM ports may be different in other machines.

8.6.2.3 CPA_DISCRETE

1. Copy the CPA_Discretes_CTA2079 folder to C:\rw_apps.
2. Create a desktop shortcut to C:\rw_apps\CPA_Discretes_CTA2079\bin\cpa_disc_cta2079
3. For Left DLCA(COM25), Click ONCE on the EDS_TAM_DISC DUAL short cut from the desktop



4. For Right DLCA(COM13), DOUBLE Click on the EDS_TAM_DISC DUAL short cut from the desktop



8.6.2.4 TTL files for Target

A_PM_CREATE.ttl

This is a teraterm macro file to send start process commands using teraterm communication port to LEFT PM.

A_PM_KILL.ttl

This is a teraterm macro file to send kill process commands using teraterm communication port to LEFT PM.

B_PM_CREATE.ttl

This is a teraterm macro file to send start process commands using teraterm communication port to RIGHT PM.

B_PM_KILL.ttl

This is a teraterm macro file to send kill process commands using teraterm communication port to RIGHT PM.

L_DLCA_CREATE.ttl

This is a teraterm macro file to send start process command using teraterm communication port to LEFT DLCA.

L_DLCA_KILL.ttl

This is a teraterm macro file to send kill process command using teraterm communication port to LEFT DLCA .

L_DLCA_NVM_PURGE.ttl

This is a teraterm macro file to send command nvm_purge using teraterm communication port to LEFT DLCA.

R_DLCA_CREATE.ttl

This is a teraterm macro file to send start process command using teraterm communication port to RIGHT DLCA

R_DLCA_KILL.ttl

This is a teraterm macro file to send kill process command using teraterm communication port to RIGHT DLCA.

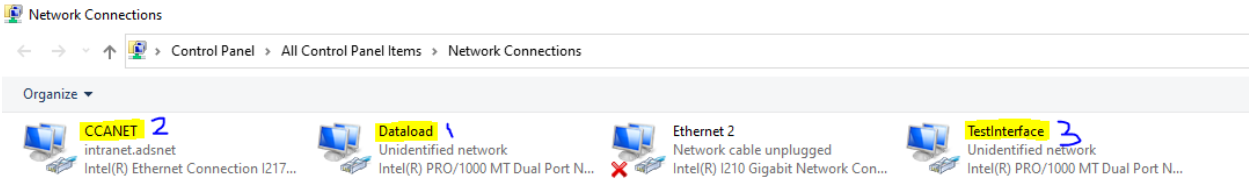
L_DLCA_NVM_PURGE.ttl

Teraterm This is a teraterm macro file to send command nvm_purge using teraterm communication port to LEFT DLCA

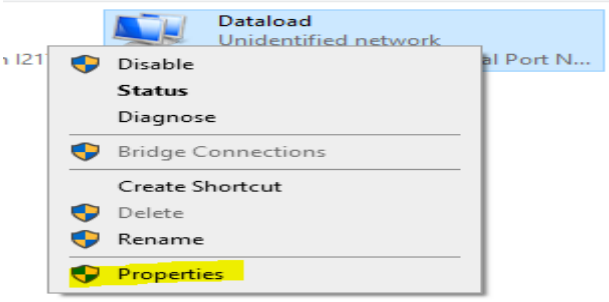
8.6.2.5 Network Connections

1. Set the'Interfacte Metrix' Value for Dataload Interface as '5' by performing the below steps

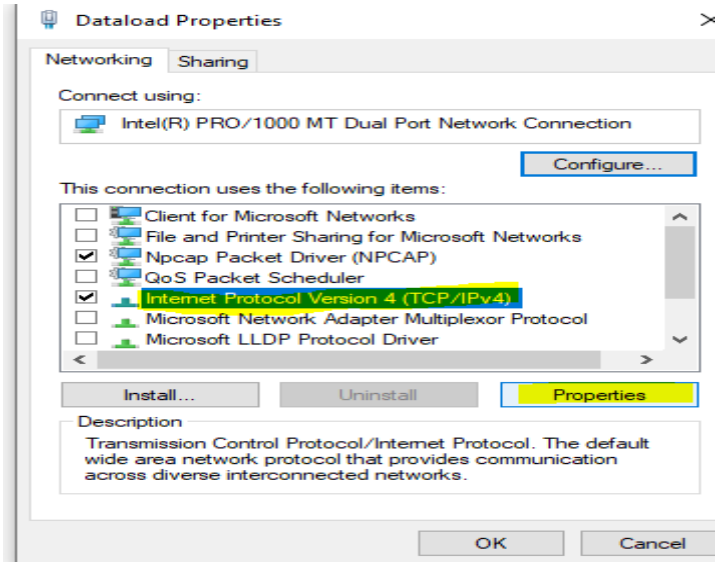
   1.1. Select 'Dataload' under Network Connections



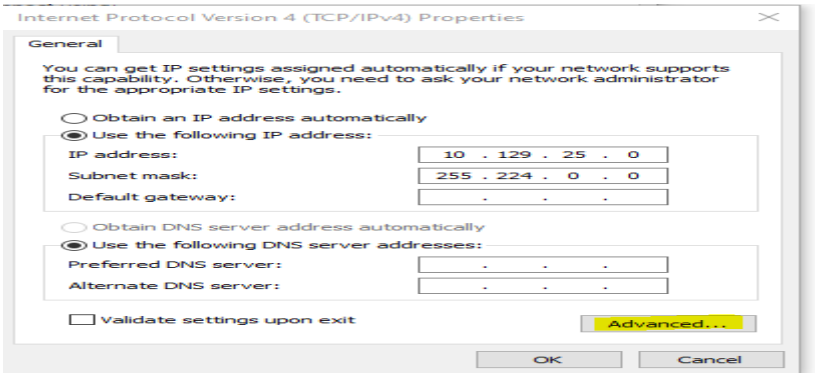
   1.2. Right click on the Dataload interface and select 'Properties'



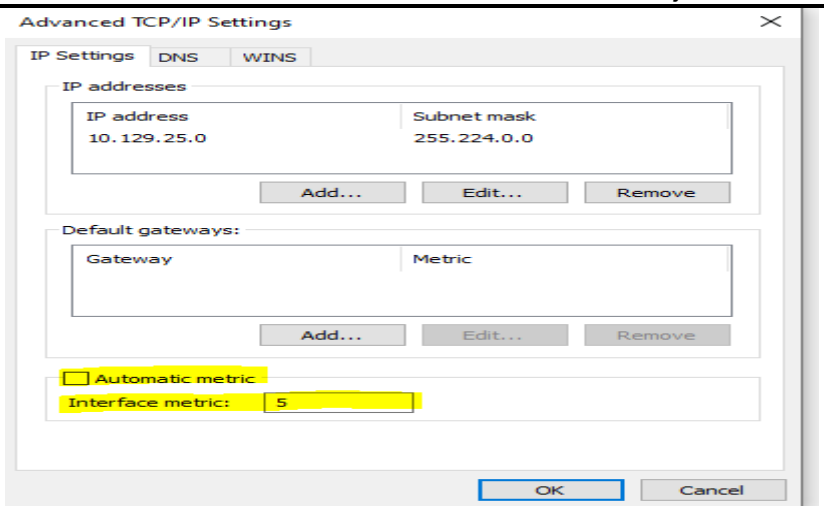
1.3. Select 'Properties' under the 'Internet Protocol Version 4(TCP/IPv4)



1.4. Select 'Advanced…' button



1.5. Uncheck the 'Automatic Metric' and Update the Interface metric value as '5'

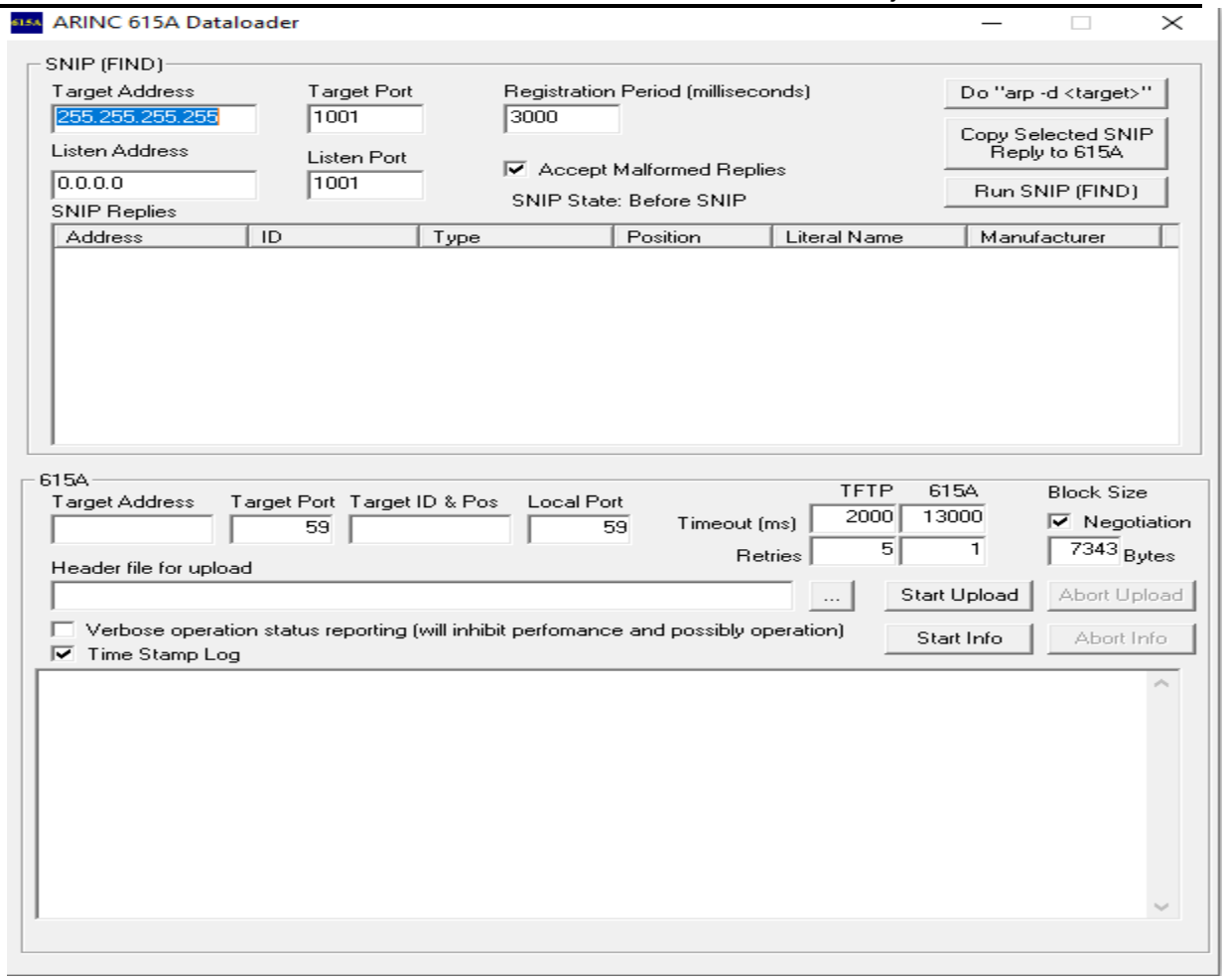


2. Set 'Interface Metric' value for CCANET/Alliance Interface as '15' by following step 2 for CCANET interface.
3. Set 'Interface Metric' value for Test Interface as '10' by following step 2 for Test Interface.

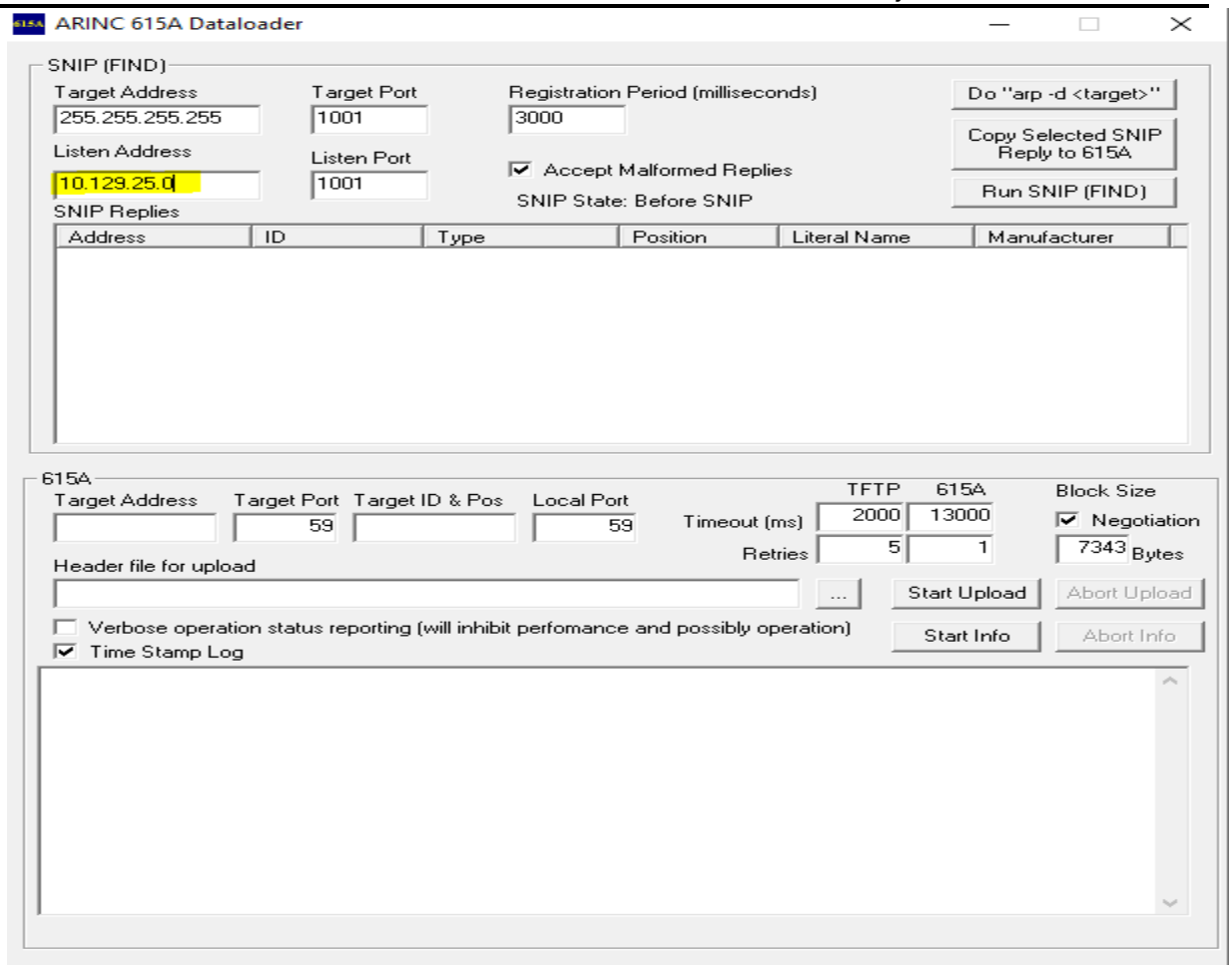
8.6.3 615A dataload

1. To perform a 615A load, a target build will need to have been successfully completed as per the guidance in the section 6.4 EDS Target Build, for EDS Target Build; and in the section 6.5 EDS Target Build with Invalid XML files, for EDS custom build. Copy the media set which need to be loaded into the EDS Target PC.
   Note: For dry-runs and RFS, get the latest builds from
   For normal build/custom build: <Dev-Branch>\Build\releases,
   For Timing build: <Verification-Branch>/test_procedures/TIMING/Mediasets/x.x.x
   where x.x.x is latest build version folder
2. Start the 615A_dataloader.exe from below svn path,

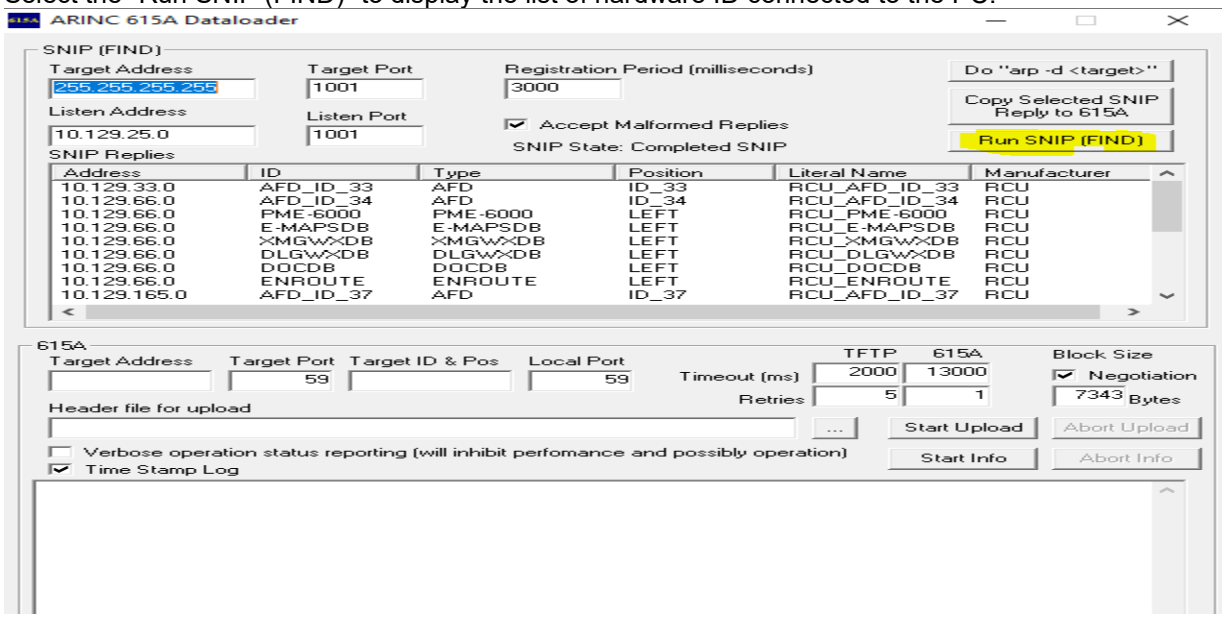
https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/tools/Target_Tools/A615_Dataloader/615A_Dataloader.exe.



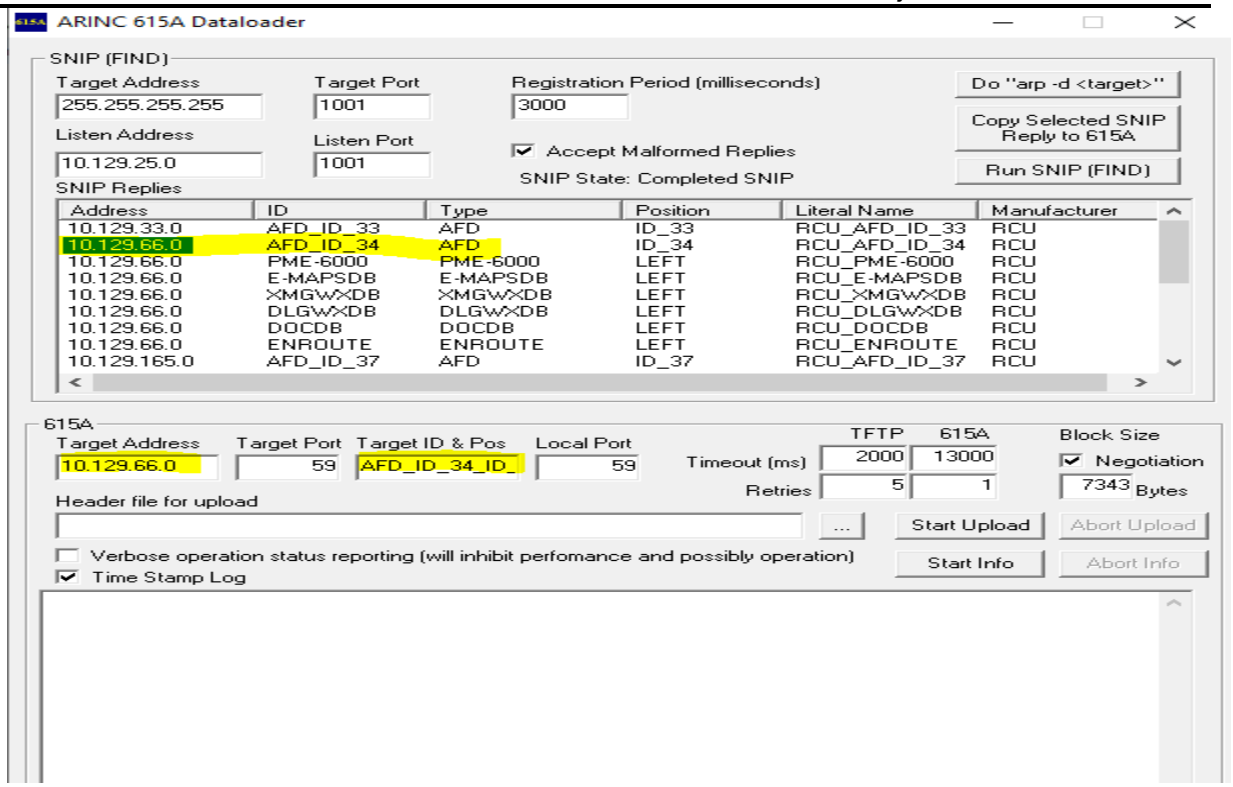
3. Update Listen Address as "10.129.25.0" (Dataload interface IP address).



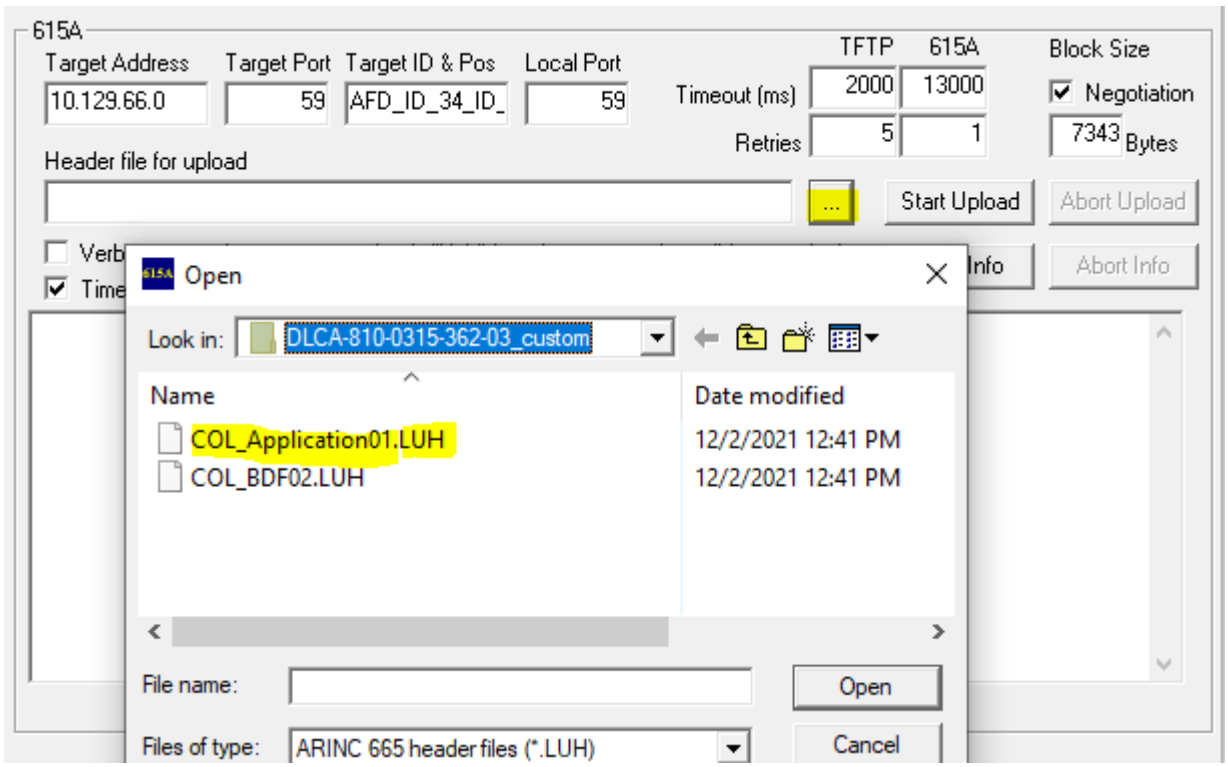
4. Select the "Run SNIP (FIND)" to display the list of hardware ID connected to the PC.



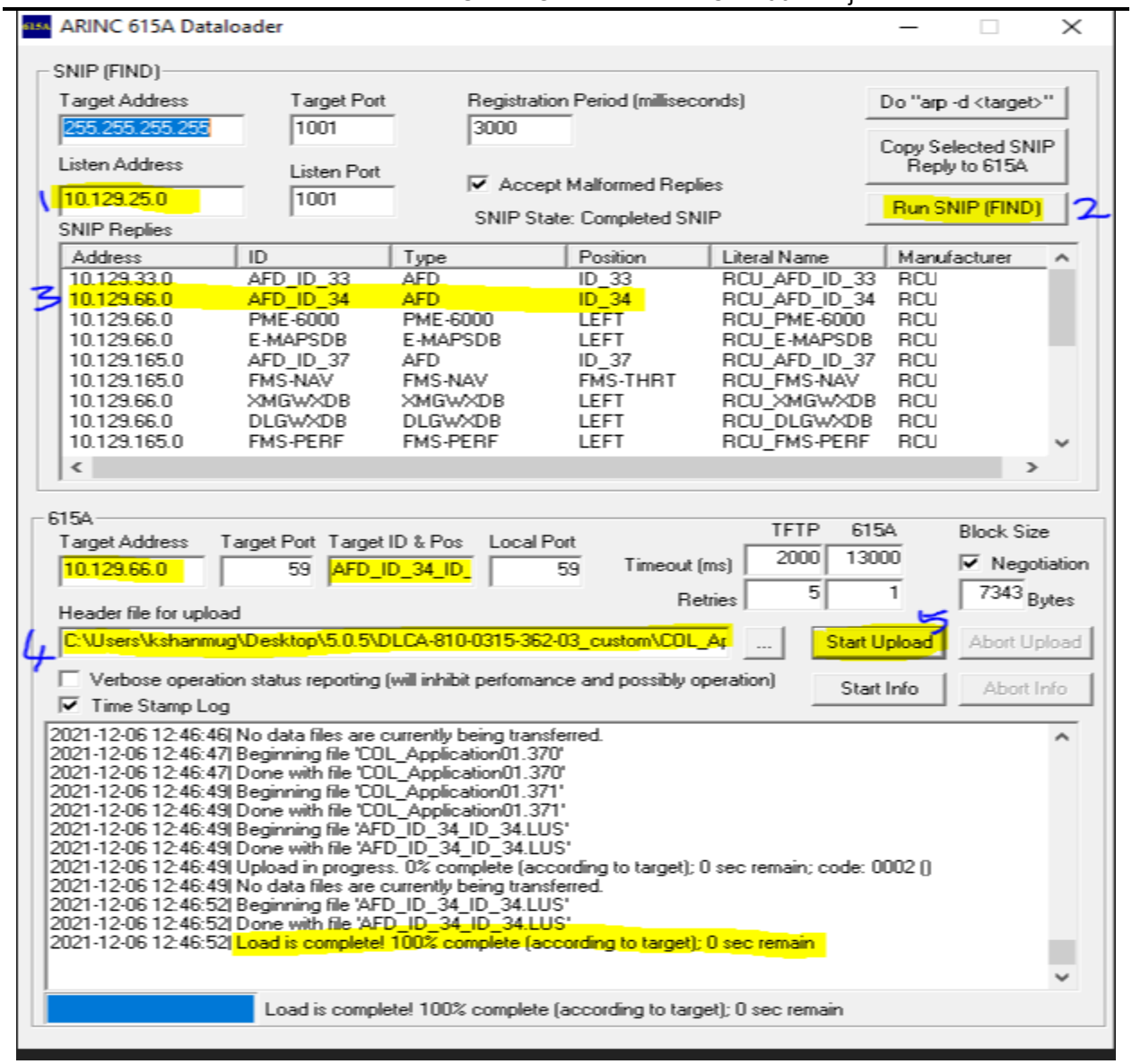
5. Select "AFD_ID_34" Id (Left AFD) from the listed hardware ID's name.



6. Select the header file which need to be loaded by selecting browse option.



7. And then press "Start Upload" to start the dataload operation,



8. Load is complete! 100% complete (according to target) message will be displayed when the dataload is completed fully
9. Repeat the steps from 13 to 16 for AFD_ID_66" Id (Right AFD)
10. Start the target. Follow the instructions as per the below section "Starting Target"

8.6.3.1 Starting Target

1. Start the VISTA environment in the https://asvn/csdlnkver-dlca-a661/branches/x.x.x_Ver/vista_sim_EDSDual/SystemTarget.bat directory of SVN to start all of the hosted applications that connect to DLCA and PM for both Left and Right sides.
2. Check for DLCA and PM Health Status for Left and Right sides. If it is off , Terminate the Vista environment and re-run A_PM_create.ttl, B_PM_CREATE.ttl, L_DLCA_CREATE.ttl and R_DLCA_create.ttl files

8.6.4 USB Dataload

USB load method is alternate method for loading on EDS Target

1. Copy all the media set files from the below svn path to the USB drive root path.

<Dev-Branch>/Build/EDS_Mediaset/MediaSet_M204_Dual/

2. Replace folder "DLCA-810-0315-470-06" in the USB, with latest EDS build from <Dev-Branch>/Build/releases
3. Insert the USB drive into the EDS Target test station.
4. Set the discrete pins as per below for doing the dataload on Left DLCA,

FL - UP
STT - UP
STL - UP
OG - UP
SSP - MIDDLE
SS2 - MIDDLE
SS1 - MIDDLE
SS0 - UP

5. Login to the CRL27090 PC which is connected to the EDS Target station.
6. Start the TAM_discretes tool from the desktop.
7. Start the P1_Com1, P2_Com1, P3_Com3, P4_Com1 and P5_Com1 teraterm window from the desktop.
8. Power cycle the Target station (Either through power switch or through 'Power on' discrete from the TAM_discretes tool)
9. After the successful dataload complete "Safe to remove USB" message will be displayed in the AFD display.
10. Set the discrete pins as per the below after doing dataload,

FL - MIDDLE
STT - UP
STL - UP
OG - UP
SSP - MIDDLE
SS2 - MIDDLE
SS1 - MIDDLE
SS0 - UP

11. Set the discrete pins as per below for doing the dataload on Right DLCA,

FL - UP
STT - UP
STL - UP
OG - UP
SSP - MIDDLE
SS2 - MIDDLE
SS1 - UP

SS0 - MIDDLE

12. Start the TAM_discretes tool from the desktop.
13. Start the P1_Com1, P2_Com1, P3_Com3, P4_Com1 and P5_Com1 teraterm window from the desktop.
14. Power cycle the Target station (Either through power switch or through 'Power on' discrete from the TAM_discretes tool)
15. After the successful dataload complete "Safe to remove USB" message will be displayed in the AFD display.
16. Set the discrete pins as per the below after doing dataload,

    FL - MIDDLE

    STT - UP

    STL - UP

    OG - UP

    SSP - MIDDLE

    SS2 - MIDDLE

    SS1 - UP

    SS0 - MIDDLE

17. Run the below command in the Processor 2 teraterm(P2_Com1) window and make sure part numbers are correct or not.

    /mnt/dlca/dlca.stripped

18. Ensure DLCA is up and running, by following steps in section 8.6.3.1 Starting Target.

9 Peer Review Preparation

Peer Reviews are held for Requirements, Design, Code, Test, as well as documents. This section gives specifics on doing a review in PREP.

Peer Reviews are done using Peer Review Eclipse Plug-in (PREP) tool, and all the review comments are maintained in PREP.

PREP Usage:

1. Welcome screen: This is PREP startup look. To close the Welcome Screen, click X. To restore it, click on Help -> Welcome

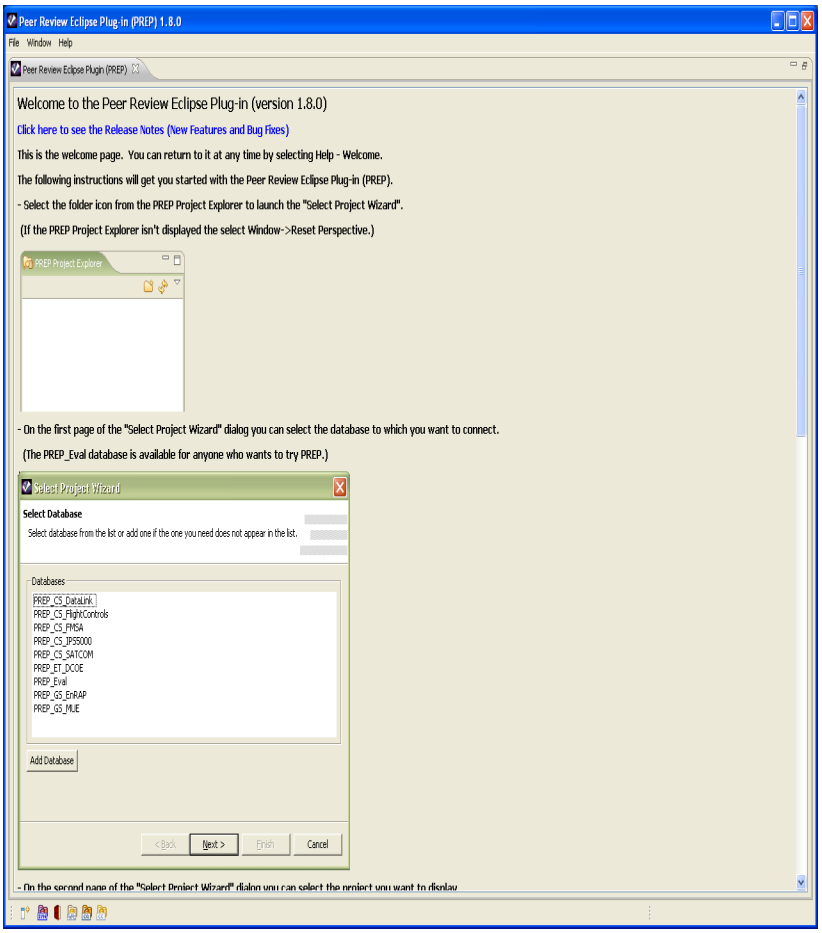


Figure 54: Peer Review Eclipse Plug-in Tool

2. PREP Findings will be implemented when state of Peer review findings are in Assigned state

   Steps to follow:

   - Click on the respective Peer Review ID for the Findings to implement.
   - Click on the Verification Tab of PREP.
   - Click on Finding Description '+ View' check box to view the findings.
   - Once the Implementer implements the finding click on Resolution '+ Edit' and provide the Changes done on Test artifacts as part of Review Findings.
   - Once Implementation is completed change the state to 'Implemented' in status tab.
   - Once the Verifier completes the Verification of Comments, Peer review finding status will be moved to closed state.

Below is the snapshot of PREP when Verification Prep ID is in State of Assigned.

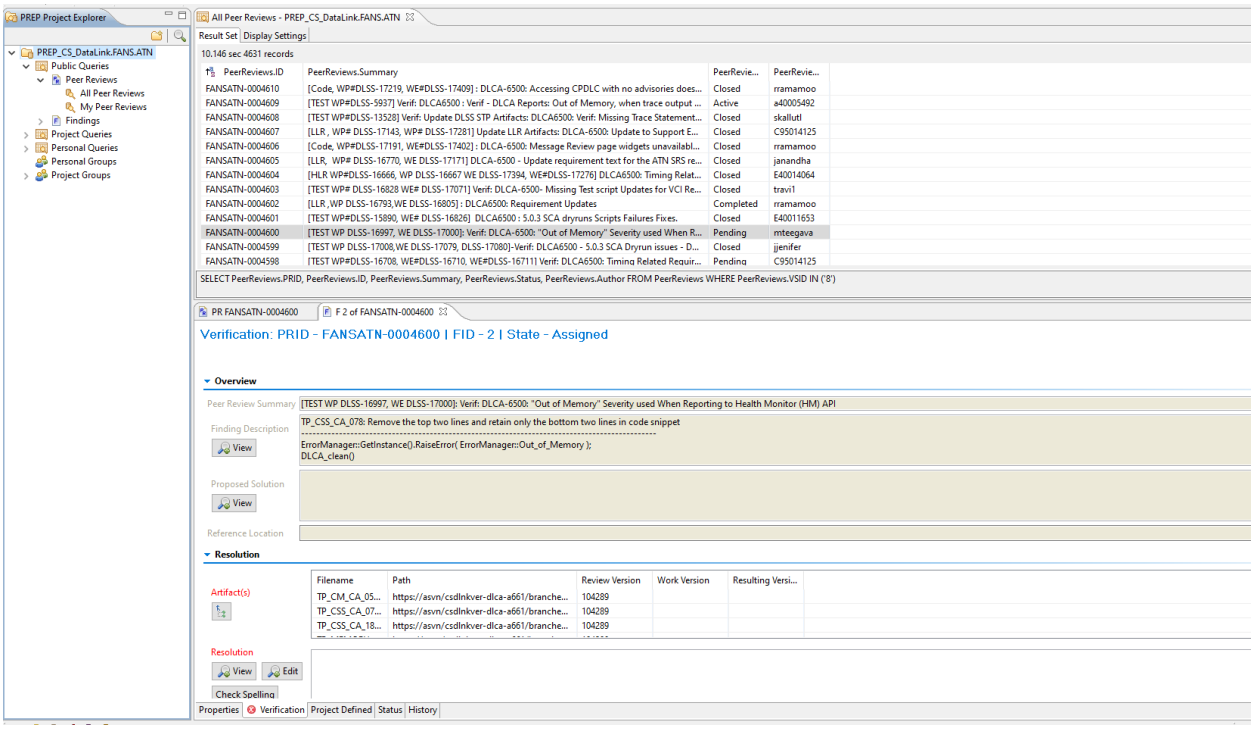


Figure 55: Example of PREP window with Prep ID

9.1 Requirements Peer Review Contents

See the Project's Software Development Plan[7]

9.2 Design Peer Review Contents

See the Project's Software Development Plan[7]

9.3 Code Peer Review Package Contents

See the Project's Software Development Plan[7]

9.4 Test Case Peer Review Package Contents

PREP peer review with

- Summary:
  [Test Case] Build 1.5.0: Review for XXXXX XXXX Page
- Description:

  This Peer Review will cover the following XXXXX XXXX requirements tested in respective Test Scripts.

  <Test Script #1>
  <Added>: HMICXX01, HMICXX02, CSSXX01
  <Modified>: HMICXX03, CSSXX02
  <Deleted>: HMICXX04

  <Test Script #2>
  <Added>: HMICXX01, CSSXX01, CSSXX02
  <Modified>: HMICXX03, CSSXX03

  STP Linking:

  TCS:

  JIRA WP:
- SRS (Baselined Version) which is being tested needs to be added in the PREP->Reference tab->Documents section to show which baselined version of SRS(s) is being tested
- SVN History for proof of SVN configuration control
  - STP (Test Case) to Doors Requirements Traceability
  - Python test case file
- Completed Test Case Review Checklist available in PREP
- For HMI pages, an updated TP_CPDLC_HMI_001.py or equivalent file for checking default page image. This file update should be limited to header test case only.
- Other files that will be used in the Test Procedure, such as image files
- Participants should include developers, SQE's, and verification team members
- The PREP review summary should contain the keywords: [Test Case]Review"

9.5 Test Procedure Peer Review Package Contents

The Test Peer Reviews are performed using PREP (Peer Review Eclipse Plug-in) and below are the artifacts used while performing the review:

- JIRA WP, if applicable
- SVN History for proof of SVN configuration control
- Python test procedure files (.py files)
- Test Procedure actual results (.log,.csv files)
- Test Case Summary (TCS) Table (.xlsx files)
- Completed Test Procedure Review Checklist available in PREP
- Other files that are used in the Test procedure, such as python utilities, A661 parser utility, custom configuration XML files are reviewed.

Note: When creating this test procedure it may determine that the test case must change, in this case the review summary should contain the keywords "[Test Procedure] Review" and developers should be invited to the review. On the other hand if the test case did not change, developers do not need to be invited to the review and the summary should contain the keywords "[Test Procedure] Review"

How to create a peer review?

Each tab in peer review has been addressed below:

Properties:

- Summary: Give a meaning full summary (prep name) for the review.

  If more than one review is scheduled for a section then set numbers should be mentioned.

  This needs to be in the following format:

  In case the test case is modified this change can be included in the summary.

  [Test Procedure] Build [build_id]: Section [section_id] [section_name]

- Description : Give a brief description of the test procedures

  This should be in the following format:

  This Peer Review will cover the Test Procedure for following sections:

  SRS for XXX - Section [xxx] [section_name]

  #TP_XXX.py:

  Added : [requirements (reqs_id) associated with script]

  Modified: [requirements (reqs_id) that are modified]

  Deleted: [requirements that are deleted]

  Test Case Modification:

  [Describe if the test case has been modified]

  JIRA WPs*:*

  ---

  1. JIRA WP : [Description]

- Review Type:
  - Select one of the review type with Desk review or Meeting review
  - Select meeting review for new scripts being added to subversion for which TPs are updated for the first time
  - Select desk review if <10% if any existing script is modified

- Detection Activity:
  Select: E-030-070 Develop Software Verification Procedures

- Network Activity:

Select the working NWA

Participants:

Select the appropriate date, time by when review should be completed. Make sure at least 2 days are given for the participants to review the artifacts.

Add required participants for the review. It should consist of producer, software reviewer, verification reviewer, quality, leader; Main Reviewer is also invited for TP reviews, who has to fill the Main reviewer check lists.

Artifacts:

- Artifacts Type:
  Select *Test Procedures and Results*
- Artifacts:
  Add .py script, .log and tcs_dlca_a661_idc.xls (Before adding this xls update all the columns associated with the script)

References:

- JIRA WPs:
  Update the verification WE associated with the WP and add to peer review
- Documents:
  Add DOORS module associated with the TPs

Checklists:

- 3 checklists gets automatically added to each review:
  1) _Leader Checklist
  2) _Test Procedures and Results – Producer
  3) _Test Procedures and Results – Reviewer
- Test Case check-lists are also need to be included under checklist tab, if there are any logical changes in the test case

As a part of checklist, we need to fill WP disposition.

- Checklists should be filled by the author of the review (also the producer) before moving the prep to pending state.

After all the required tabs have been filled peer review gets created!

10 Verification Method Selection Guidance

Guidance on which verification method to select is based on team member concurrence with the following DO-178B guidelines (which follow).

- The criticality of the software requirement.
- Effort associated with the development of the verification method.
- The resulting quality of the verification method in achieving verification of the software requirement.

Refer RTCA document: RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification, December 1, 1992.

The following are applicable DO-178B guidelines for selecting a verification method:

6.2.d, "When it is not possible to verify specific software requirements by exercising the software in a realistic test environment, other means should be provided and their justification for satisfying the software verification process objectives defined in the Software Verification Plan or Software Verification Results."

6.4, "Testing of airborne software has two complementary objectives. One objective is to demonstrate that the software satisfies its requirements. The second objective is to demonstrate with a high degree of confidence that errors, which could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed.

The objectives of the three types of D0-178B testing are:

- Hardware/software integration testing: To verify correct operation of the software in the target computer environment.
- Software integration testing: To verify the interrelationships between software requirements and components and to verify the implementation of the software requirements and software components within the software architecture.
- Low-level testing: To verify the implementation of software low-level requirements.

Note: *If a test case and its corresponding test procedure are developed and executed for hardware/software integration testing or software integration testing and satisfy the requirements-based coverage and structural coverage, it is not necessary to duplicate the test for low-level testing. Substituting nominally equivalent low-level tests for high-level tests may be less effective due to the reduced amount of overall functionality tested."*

6.4.1, "More than one test environment may be needed to satisfy the objectives for software testing. An excellent test environment includes the software loaded into the target computer and tested in a high fidelity simulation of the target computer environment.

Note: *In many cases, the requirements-based coverage and structural coverage necessary can be achieved only with more precise control and monitoring of the test inputs and code execution than generally possible in a fully integrated environment. Such testing may need to be performed on a small software component that is functionally isolated from other software components.*

Certification credit may be given for testing done using a target computer emulator or a host computer simulator. Guidance for the test environment includes:

Selected tests should be performed in the integrated target computer environment, since some errors are only detected in this environment."

6.4.3, "Requirements-based testing methods consist of methods for requirements-based hardware/software integration testing, requirements-based software integration testing,

and requirements-based low-level testing. With the exception of hardware/software integration testing, these methods do not prescribe a specific test environment or strategy. Guidance includes:

a) Requirements-Based Hardware/Software Integration Testing: This testing method should concentrate on error sources associated with the software operating within the target computer environment, and on the high-level functionality. The objective of requirements-based hardware/software integration testing is to ensure that the software in the target computer will satisfy the high-level requirements.

   Typical errors revealed by this testing method include:

   - Incorrect interrupt handling.
   - Failure to satisfy execution time requirements.
   - Incorrect software response to hardware transients or hardware failures, for example, start-up sequencing, transient input loads and input power transients.
   - Data bus and other resource contention problems, for example, memory mapping.
   - Inability of built-in test to detect failures.
   - Errors in hardware/software interfaces.
   - Incorrect behavior of feedback loops.
   - Incorrect control of memory management hardware or other hardware devices under software control.
   - Stack overflow.
   - Incorrect operation of mechanism(s) used to confirm the correctness and compatibility of field-loadable software.
   - Violations of software partitioning.

b) Requirements-Based Software Integration Testing: This testing method should concentrate on the inter-relationships between the software requirements, and on the implementation of requirements by the software architecture. The objective of requirements-based software integration testing is to ensure that the software components interact correctly with each other and satisfy the software requirements and software architecture. This method may be performed by expanding the scope of requirements through successive integration of code components with a corresponding expansion of the scope of the test cases.

   Typical errors revealed by this testing method include:

   - Incorrect initialization of variables and constants.
   - Parameter passing errors.
   - Data corruption, especially global data.
   - Inadequate end-to-end numerical resolution.
   - Incorrect sequencing of events and operations.

c) Requirements-Based Low-Level Testing: This testing method should concentrate on demonstrating that each software component complies with its low-level requirements. The objective of requirements-based low-level testing is to ensure that the software components satisfy their low-level requirements.

   Typical errors revealed by this testing method include:

   - Failure of an algorithm to satisfy a software requirement.
   - Incorrect loop operations.

- Incorrect logic decisions.
- Failure to process correctly legitimate combinations of input conditions.
- Incorrect responses to missing or corrupted input data.
- Incorrect handling of exceptions, such as arithmetic faults or violations of array limits.
- Incorrect computation sequence.
- Inadequate algorithm precision, accuracy or performance.

11 Software Verification Procedures and Results Document & CPCI

Refer RTCA document: RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification, December 1, 1992

DO-178B guidelines for Section 11.14, Software Verification Results are repeated here for guidance on the Software Verification Procedures and Results (SVPR)[8]. The Software Verification Results are produced by the software verification process activities.

Software Verification Results should be as below:

a. For each review, analysis and test indicate each procedure that passed or failed during the activities and the final pass/fail results.

b. Identify the configuration item or software version reviewed, analyzed or tested.

c. Include the results of tests, reviews and analyses, including coverage analyses and traceability analyses.

The SVPR Document and associated CPCI will be released at the conclusion of the verification effort. The CPCI will be a Level 1, no re-build, release. Only Peer Reviewed and approved verification artifacts will be contained in the SVPR Document and CPCI. The SVPR Document primarily includes summarized verification activities performed. The SVPR CPCI primarily is for collecting electronic copies of the verification artifacts. A well thought out and coordinated baseline SVPR Document and CPCI will assist greatly in future project re-verification efforts, especially when only incremental verification is needed.

The SVPR Document will contain the following:

- A summary of all Peer Reviews
- A summary of all analysis procedures (including a description of exact methods) and results
- The requirements traceability matrix will be included as an appendix
- A directory path to the summary of all Test Procedure documents (i.e. TCS tables)
- A special section on the partitioning verification activity of the user-modifiable software

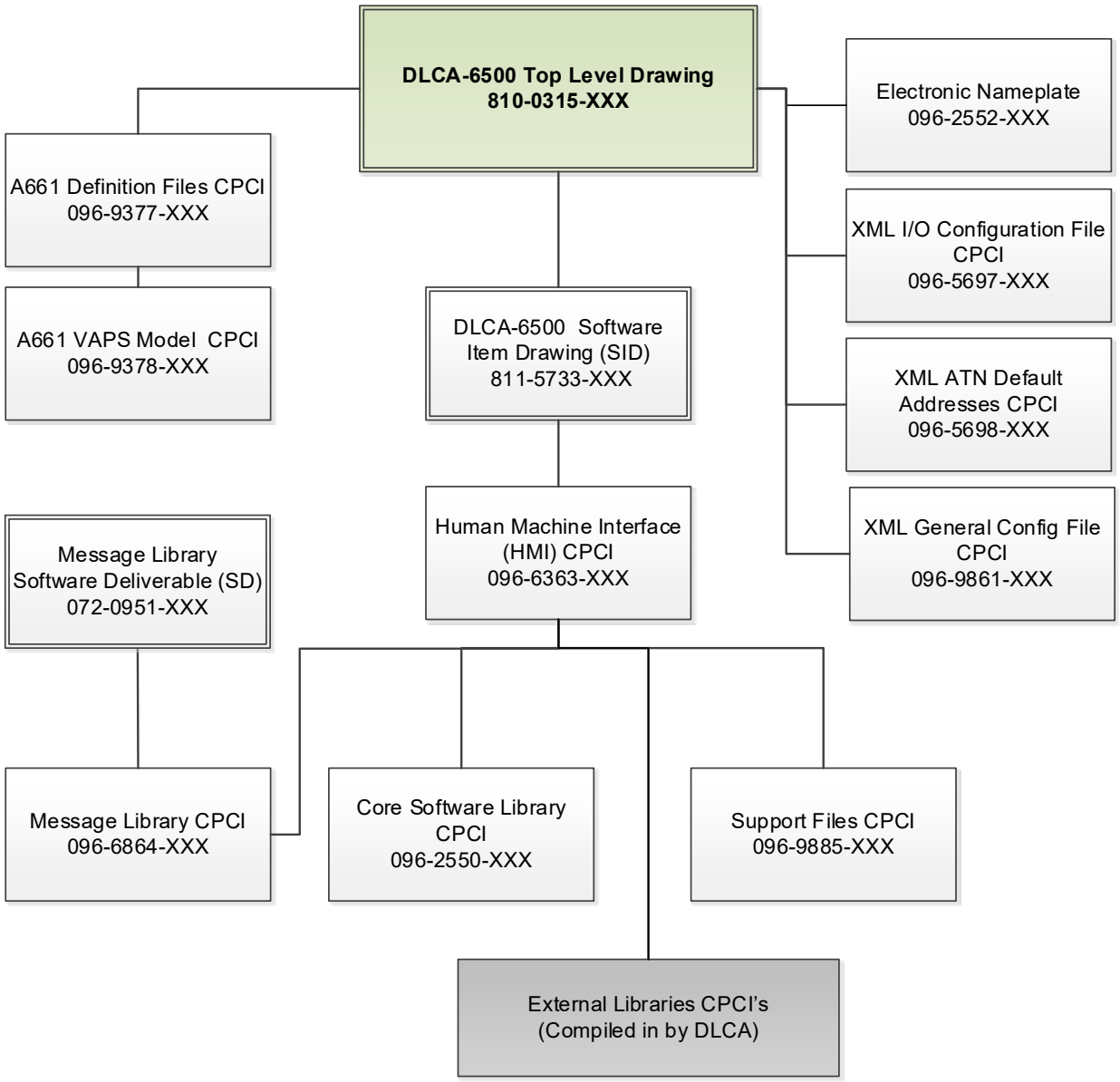
The SVPR CPCI will contain the following:

- The structural coverage results database
- The Test Procedure document files
- The Test Procedure executable test program files
- The actual test/verification results files
- The TCS table files

At the conclusion of the verification effort a "version label" will be placed on all the associated verification artifacts included in the SVPR CPCI.

Refer to the below DLCA 6500 CPCI framework to understand how the CPCI modules are partitioned.

DLCA 6500 CPCI framework



12 Execution tools

For executing multiple files at single time, we use execution tools like Batch execution tool and DVM tool

12.1 Batch Execution tool – Method 1

Location: The Subversion directory associated with Batch Execution tool is located in the SVN repository path /tools/Batch_Execution

Use the following steps to execute the scripts:

1. Run the DLCA_BatchExecution.pyw
   (Double click it if python is installed successfully we will get a GUI)

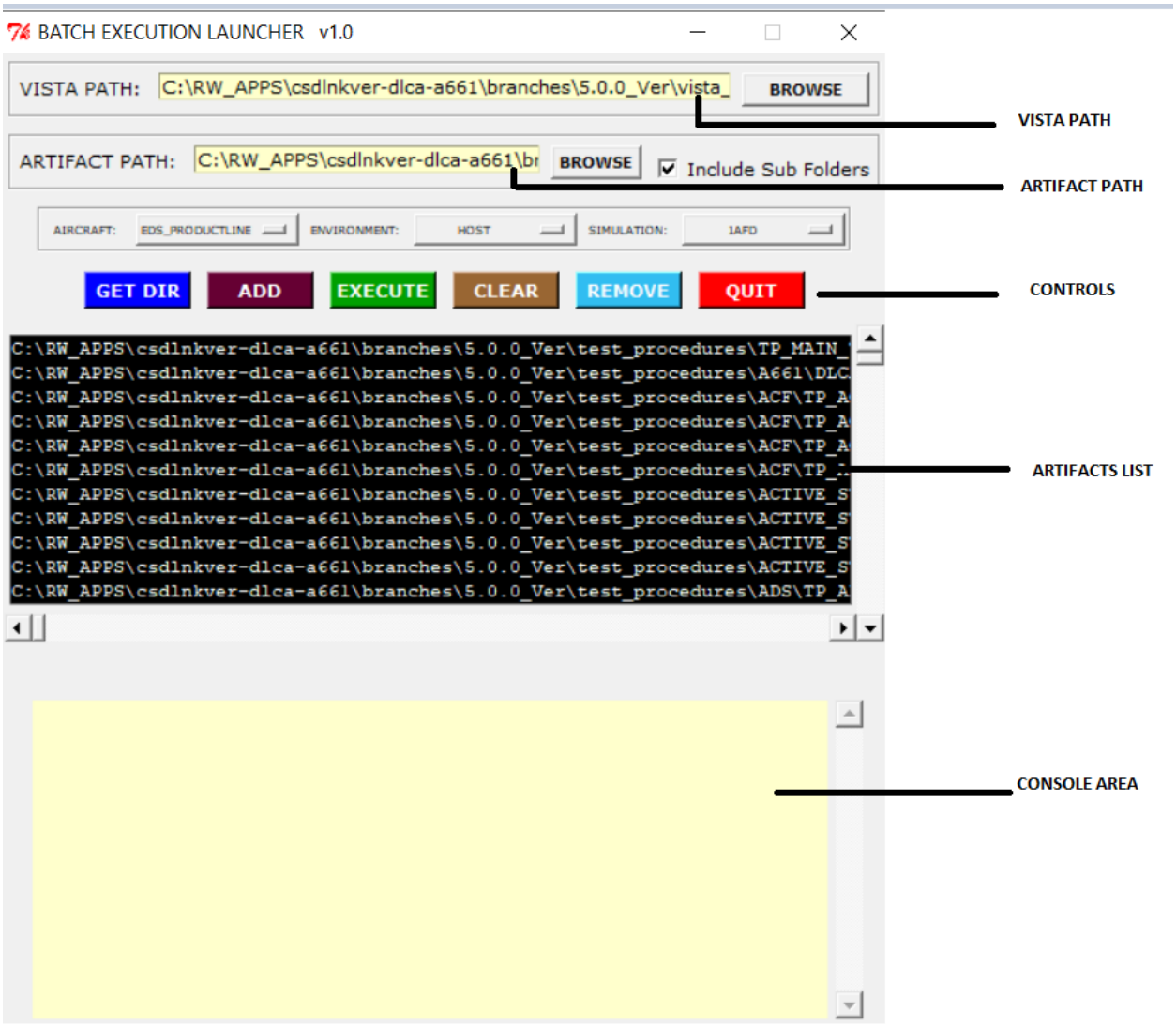


Figure 56: Batch Execution tool

2. Select *VISTA PATH*, generally it will be present at following path
   vista_sim/Vista_tools/vista/system/launch_vista.exe -script=python275 –wait

Note: "-script=python275 -wait" statement needs to be appended with vista path to support commands written in Python 2.7.5.

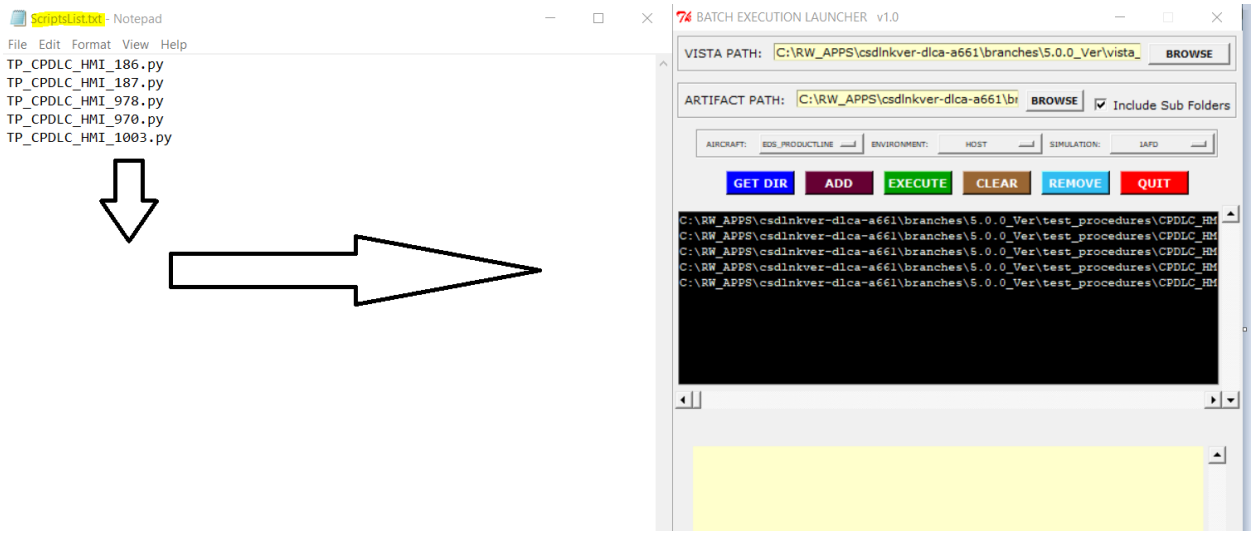
3. Select *ARTIFACT PATH*, which is nothing select the folder where test scripts are present to be execute if you want to include subfolders also select '*Include Sub Folders*'
4. If valid path selected in for *ARTIFACT PATH,* we will get all python files present in selected folder in Artifacts area of batch execution tool.
5. Select the files, which ever want to execute and then select ADD control to add them to queue, then we will get status message in below console area that respective files are added
6. On completion of adding files to queue, select EXECUTE control, tool will start the execution of selected test scripts in order
7. If execution of all scripts added to queue is completed then we will get message in console stating *"Execution Completed…"*
8. To clear the queue, select the CLEAR control
9. Repeat the steps from 4 to add files and to execute.
10. Log files for the executed test scripts can be found in the same folder as the executed test script.

12.2 Batch Execution tool – Method 2

Location: The Subversion directory associated with Batch Execution tool is located in the SVN repository path /tools/BatchExecution_Method2

Method 2 is another version of Batch Execution tool, Where the list of artifacts in the ARTIFACT LIST area can be controlled from an external .txt file called ScriptsList.txt (Location: /tools/BatchExecution_Method2/ScriptsList,txt)

The Artifacts List area will only display the scripts that are listed in ScriptsList.txt. Rest of the execution steps are similar to Method 1. This facility is added to avoid the scroll and search functionality. Now the user can select all the scripts using ADD control to add them to queue,



12.3 DVM tool

DVM application works in client server configuration. 1 PC should be configured as server and the other PC's are connected clients. Location: <Verification-Branch>/tools/DVM_simplified.

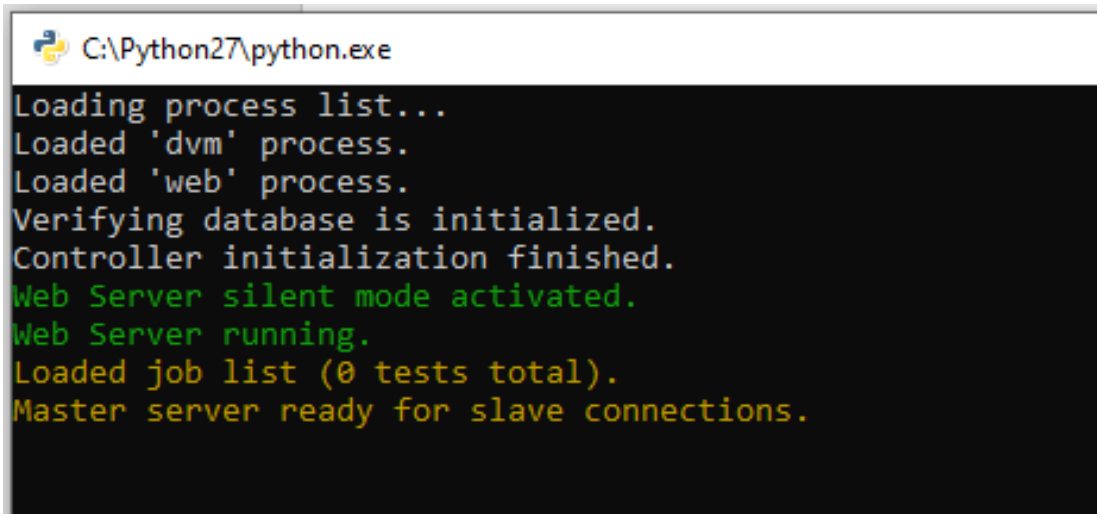
Note: <Verification-Branch> refers to: for verification development activities: https://alasvn/csdlnkver-dlca-a661/branches/5.0.0_Ver, for formal executions: Tag created for verification files.

All tests are added to the server PC and the tests will be executed in client PC's.

Steps for Execution Procedure:

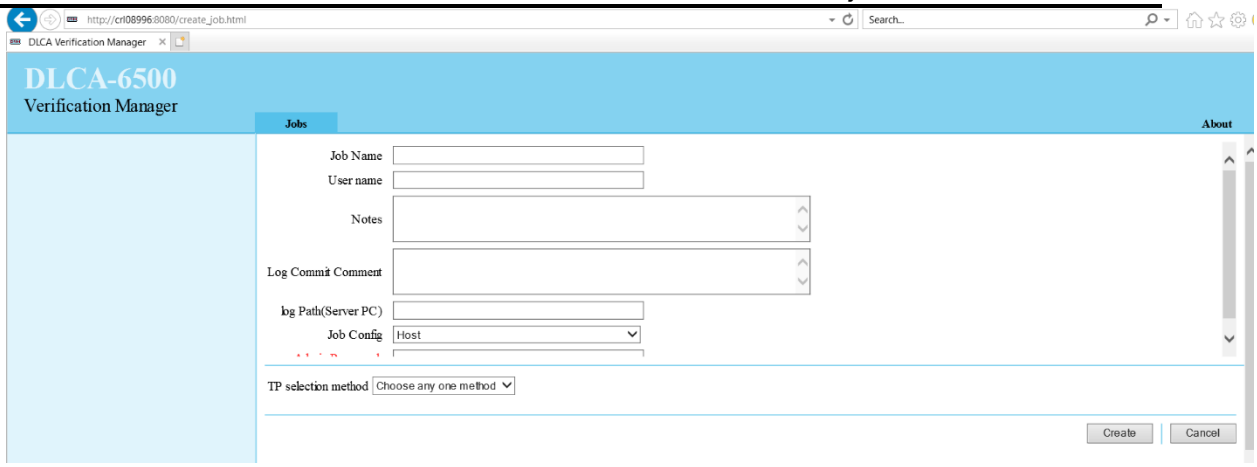
1. Perform tortoise svn clean up and remove read-only property of <Verification-Branch>/tools/DVM_simplified folder, before each instance of DVM execution.

2. For getting Master server ready for slave connections, execute the script <Verification-Branch>/tools/DVM_simplified/start.py in python, and ensure "Master server ready for slave connections." Is displayed.

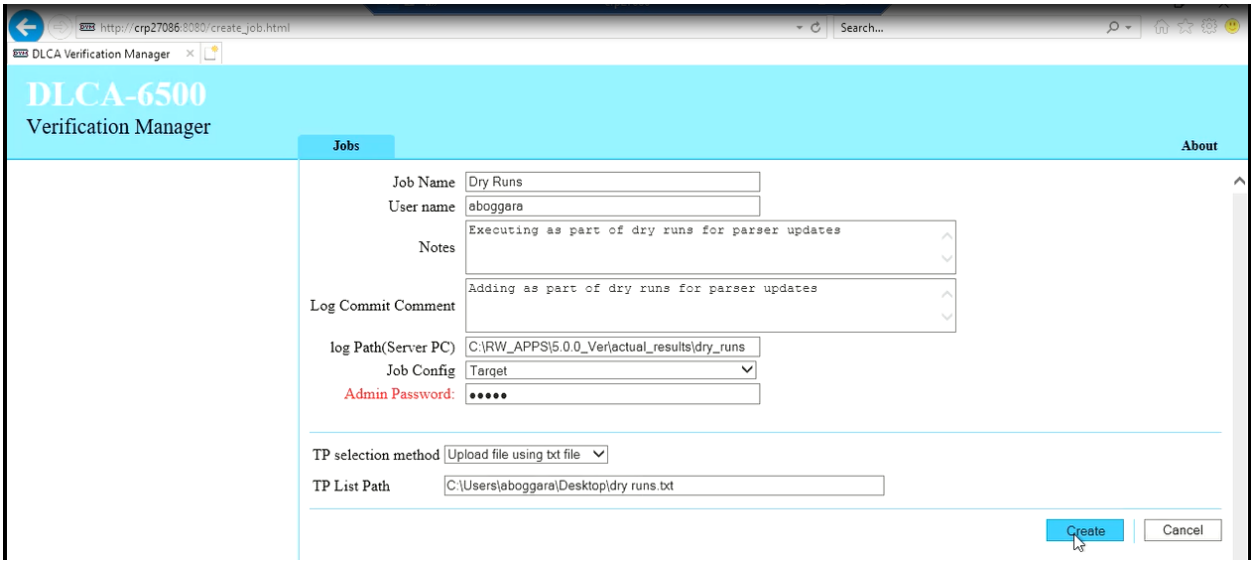


3. Open internet explorer, and enter http://CRP27086:8080

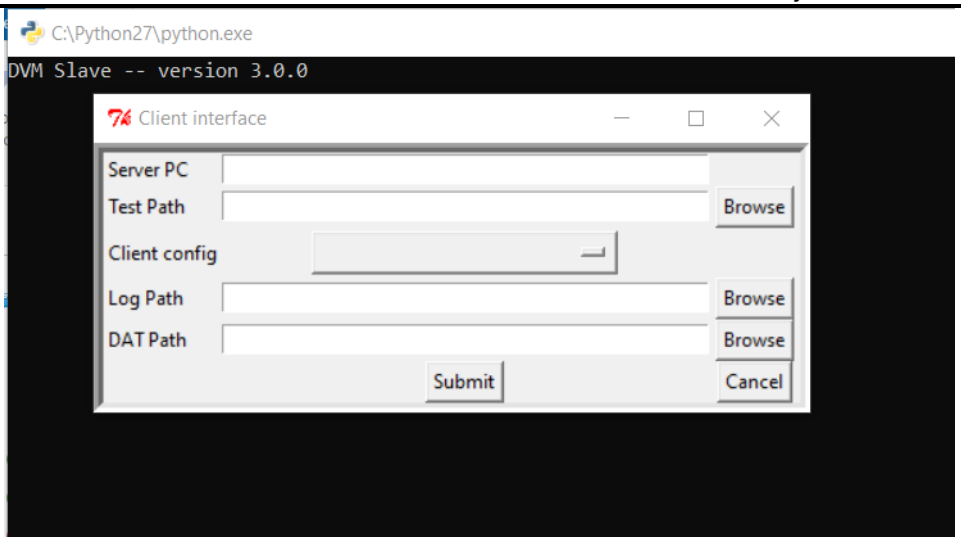
4. Click on Jobs, and select "Add Job".



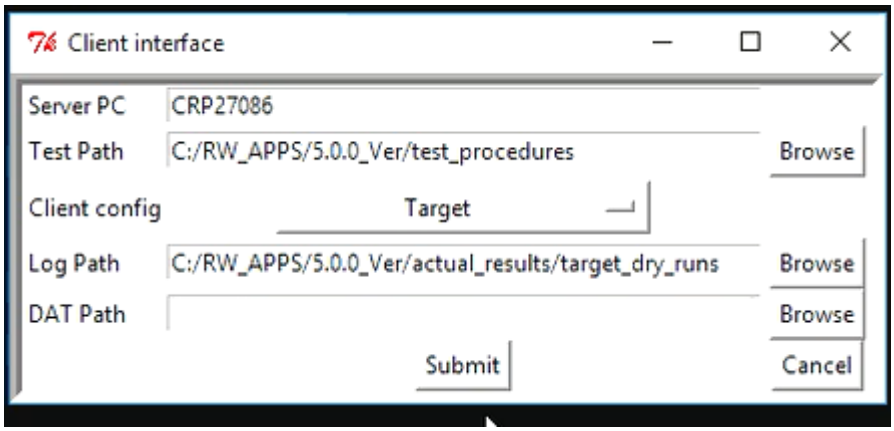
5. Enter suitable details as shown in below screen shot. Dry runs.txt file contains list of test script names for execution.[for example TP_ACF_004.py, from the path <Verification-Branch>/test_procedures/. Admin Password is admin . Select Create for the execution job to be created.



6. Execute the script <Verification-Branch>/tools/DVM_simplified/dvm_slave.py

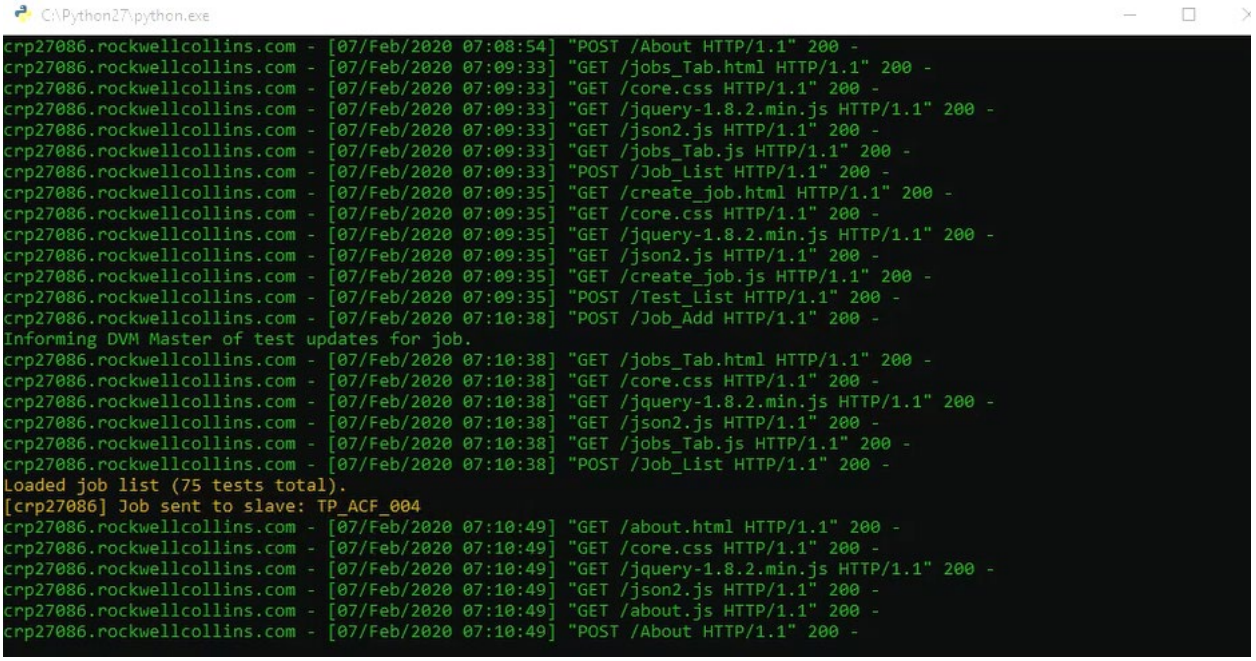


7. Enter name of Server PC, for example CRP27086. For Test Path, browse for test procedure folder for example <Verification-Branch>/test_procedures/. Select Target in Client config, for target executions; select SCA for SCA executions. For Log Path, browse for log path where results will be generated for example <Verification-Branch>\actual_results\target_dry_runs. Select Submit, and observe "waiting for job request from server" in python console in slave pc. And observe "Slave connected" message in server pc python console.



8. In DLCA-6500 Verification Manager, Click on About, after few seconds, say 20 sec, and observe status of script is running on the Slave node, as shown below.

9. Observe message "Loaded Job list" and "job sent to slave" message in python console



13 Incremental Verification

Once verification has been performed and completed (i.e. WPs required for maintenance) for the entire DLCA to DO-178B Level C, incremental verification can be established. The incremental verification effort will primarily be determined by the changes identified and their impact on the rest of the DLCA. The changes and their impact will be documented in an Impact Analysis. The Impact Analysis would start with determining the affected SRS requirements. The Impact Analysis would be the driver of the incremental verification effort. The Impact Analysis at a minimum, would justify and identify which requirements are affected, which surrounding requirements are affected, which software packages need to be Peer Reviewed, which Test Procedures need to be updated and which rerun, and what source lines of code need to have structural coverage. The results of this Impact Analysis will be documented in the Jira WPs.