# GOTO: FILE -> MAKE A COPY to save for your own use.

# 

# Section 1 - Project Description

## 1.1 Project

Elevate Oil Level Sensor Test Tank

## 1.2 Description

The Elevate Oil Level Sensor Test Tank (Elevate OLS-TT) uses LabVIEW to Automate the ATP process and add AEPS Testing functionality.

## 1.3 Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Comment** | **Author** |
| 6-10-2024 | Initial Release | R. Ales Consulting |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Contents**

[Section 1. GOTO: FILE -> MAKE A COPY to save for your own use.](#_Toc196746789)

[Section 2.](#_Toc196746790)

[Section 3. Section 1 - Project Description](#_Toc196746791)

[3.1 1.1 Project](#_Toc196746792)

[3.2 1.2 Description](#_Toc196746793)

[3.3 1.3 Revision History](#_Toc196746794)

[Section 4. Section 2 - Overview](#_Toc196746795)

[4.1 2.1 Purpose](#_Toc196746796)

[4.2 2.2 Scope](#_Toc196746797)

[4.3 2.3 Requirements](#_Toc196746798)

[Section 5. Section 3 - System Architecture](#_Toc196746799)

[Section 6. Section 4 - Data Dictionary](#_Toc196746800)

[Section 7. Section 5 - Software Domain Design](#_Toc196746801)

[7.1 5.1 Software Application Domain Chart](#_Toc196746802)

[7.2 5.2 Software Application Domain](#_Toc196746803)

[7.2.1 5.2.1 Domain X](#_Toc196746804)

[7.2.1.1 5.2.1.1 Component Y of Domain X](#_Toc196746805)

[7.2.1.1.1 5.2.1.1.1 Task Z of Component Y1 of Domain X](#_Toc196746806)

[Section 8. Hardware Design](#_Toc196746807)

[8.1 DataQ 2108](#_Toc196746808)

[8.1.1 Sample Rate](#_Toc196746809)

[8.2 Pump Controller-7](#_Toc196746810)

[Section 9. – Data Design](#_Toc196746811)

[9.1 6.1 Persistent/Static Data](#_Toc196746812)

[9.1.1 6.1.1 Dataset](#_Toc196746813)

[9.1.2 6.1.2 Static Data](#_Toc196746814)

[9.1.3 6.1.3 Persisted data](#_Toc196746815)

[9.2 6.2 Transient/Dynamic Data](#_Toc196746816)

[9.3 6.3 External Interface Data](#_Toc196746817)

[9.4 6.4 Transformation of Data](#_Toc196746818)

[Section 10. Section 7 - User Interface Design](#_Toc196746819)

[10.1 7.1 User Interface Design Overview](#_Toc196746820)

[10.2 7.2 User Interface Navigation Flow](#_Toc196746821)

[10.3 7.3 Use Cases / User Function Description](#_Toc196746822)

[Section 11. Section 8 - Other Interfaces](#_Toc196746823)

[11.1 8.1 Interface X](#_Toc196746824)

[Section 12. Section 9 - Extra Design Features / Outstanding Issues](#_Toc196746825)

[Section 13. Section 10 – References](#_Toc196746826)

[Section 14. Section 11 – Glossary](#_Toc196746827)

[Section 15. LabVIEW](#_Toc196746828)

[15.1 Merging VIs](#_Toc196746829)

[Section 16. Configuration Management](#_Toc196746830)

[16.1 Software Versioning](#_Toc196746831)

[16.2 Git Branching Strategy](#_Toc196746832)

[16.3 Setting up Git](#_Toc196746833)

[16.3.1 Step 1: Configure Git](#_Toc196746834)

[16.3.2 Step 2: Initialize a New Repository](#_Toc196746835)

[16.3.3 Step 3: Cloning an Existing Repository](#_Toc196746836)

[16.4 Pull from GitHub](#_Toc196746837)

[16.5 Building the LabVIEW App Executable](#_Toc196746838)

[16.6 Deploying the LV app to the target test-PC.](#_Toc196746839)

[16.7 Using Git Hints](#_Toc196746840)

[16.7.1 Creating a Local Project Folder](#_Toc196746841)

[16.7.2 Switching to a Remote Branch](#_Toc196746842)

[16.7.3 Switching to a Previous Commit](#_Toc196746843)

[16.7.4 Create a new branch and save to remote GitHub](#_Toc196746844)

[16.7.5 Add a Prefix to the Names of Multiple Files in Power Shell](#_Toc196746845)

# Section 2 - Overview

## 2.1 Purpose

The primary purposes of the Elevate OLS-TT are twofold. First, to automate using LabVIEW, making the current manual ATP testing method more efficient; and second, to collect and record more precise data during ATP testing.

## 2.2 Scope

The Elevate OLS-TT roadmap consists of modules of functionality to be phased in as they are completed as follows:

1. Single Sensor test with voltage divider type measurement circuit. The intent of this first implementation is to prove the concept and produce valid data for engineering evaluation.
2. Multiple sensor test with aircraft compliant type measurement circuit, and pump speed control on a custom interface board. The intent of this implementation is to be ATP compliant and approved by the customer.
3. Variations that aid in the production processes, for example, pre-potting verification of switch installation and adjustment of the PCB assembly position.
4. Modifications to the basic test protocol and equipment will be migrated to other level sensor products.

In this current writing, only the first phase with be addressed.

## 2.3 Requirements

System requirements are specified in the Requirements Document *Elevate OLS ATP Requirements 2-7-24 (003).*

# Section 3 - System Architecture

Describe/include a figure of the overall system architecture (and where this module fits in)

A diagram of a computer

Description automatically generated

# Section 4 - Data Dictionary

Brief description of each element in this module or a link to an actual data dictionary

(template of a database table description)

|  |
| --- |
| **Table** |

|  |  |  |
| --- | --- | --- |
| **Field** | **Notes** | **Type** |
| ID | Unique Identifier from TABLE\_SEQ | DECIMAL |
| NAME | The Name in Object.Name() | VARCHAR |
| VALUE | The Value output from somewhere | VARCHAR |

# Section 5 - Software Domain Design

## 5.1 Software Application Domain Chart

Describe / chart each major software application domain and the relationships between objects (UML, etc)

## 5.2 Software Application Domain

A Comprehensive high level description of each domain (package/object wherever it is better to start) within the scope of this module (or within the greater scope of the project if applicable)

### 5.2.1 Domain X

A high level description of the family of components within this domain and their relationship. Include database domain, stored procedures, triggers, packages, objects, functions, etc.

#### 5.2.1.1 Component Y of Domain X

Define Component Y, describe data flow/control at component level

##### 5.2.1.1.1 Task Z of Component Y1 of Domain X

Define Task Z, describe data flow/control at task level

# Hardware Design

## DataQ 2108

### Sample Rate

The DI-2108 maintains an internal scan list of as many as eleven different items at once. The maximum sample throughput rate is 220 kHz, 20 kHz per scan list element with all positions enabled. Analog channels support a maximum sample throughput rate of 160 kHz, one analog channel at 160 kHz, two analog channels 80 kHz per channel, etc. The scan list can be populated with any combination of analog input channels, digital input ports, the rate channel, and the counter channel.

## Pump Controller-7

# – Data Design

Describe the data contained in databases and other shared structures between domains or within the scope of the overall project architecture

## 6.1 Persistent/Static Data

Describe/illustrate the logical data model or entity relationship diagrams for the persistent data (or static data if static)

### 6.1.1 Dataset

Describe persisted object/dataset and its relationships to other entities/datasets

### 6.1.2 Static Data

Describe static data

### 6.1.3 Persisted data

Describe persisted data

## 6.2 Transient/Dynamic Data

Describe any transient data, include any necessary subsections

## 6.3 External Interface Data

Any external interfaces’ data goes here (this is for the data, section 8 is for the interface itself)

## 6.4 Transformation of Data

Describe any data transformation that goes on between design elements

# Section 7 - User Interface Design

## 7.1 User Interface Design Overview

Pictures, high level requirements, mockups, etc.

## 7.2 User Interface Navigation Flow

Diagram the flow from one screen to the next

## 7.3 Use Cases / User Function Description

Describe screen usage / function using use cases, or on a per function basis

# Section 8 - Other Interfaces

Identify any external interfaces used in the execution of this module, include technology and other pertinent data

## 8.1 Interface X

Describe interactions, protocols, message formats, failure conditions, handshaking, etc

# Section 9 - Extra Design Features / Outstanding Issues

Does not fit anywhere else above, but should be mentioned -- goes here

# Section 10 – References

Any documents which would be useful to understand this design document or which were used in drawing up this design.

# Section 11 – Glossary

Glossary of terms / acronyms

# LabVIEW

## Merging VIs

The two files to be compared must have different names.

1. Open both VIs to be merged placing them both in memory
2. On one VI, click on Menu>Tools>Merge>Merge VIs…
3. In the “Their VI” navigate to one of the files to be merged.
4. In the “Your VI” navigation to the other file to be merged.
5. In the “Base VI” navigate to the file to contain the result of the merge.

These two files will be tagged as “Theirs” and

# Configuration Management

Configuration Management practice of tracking and controlling changes to a system.

## Software Versioning

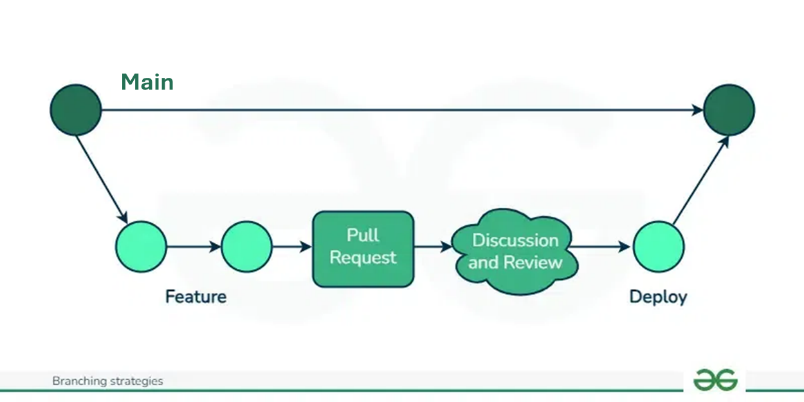
Version indicator consisting of two values the first is letter representing the Major revision and a number representing a minor revision.

* Major revision indicates the following applicability of the code:
  + Where E (Engineering) versions are for development evaluation only.
  + Where Q (Quality) versions are for interdepartmental evaluation but not for production, often referred to as Beta software.
  + Where letters A-D, F-P represent code has been qualified and released for production, the first release starts with A then incremented with a major release defined as not 100% backwards compatible (form, fit and function) with the prior release.
* Minor revision are two numbers, incremented with any change that remains 100% backwards compatible, (not materially affecting form, fit or function.)

## Git Branching Strategy

A branching strategy is a strategy that software development teams adopt for writing, merging and deploying code with the help of a version control system like Git. Since this project is likely to have a single developer the simpler, streamlined branching Github Flow strategy shall be used.

* **Main Branch** The primary branch of a Git repository where the most recent stable, production-ready code resides.
* **Feature Branch** is a development branch, created to work on a specific feature or task isolated from the main branch. Feature branch code distributed for deployment versioned as the Next release version prefixed with a Q. For example, if the Main Brance is version A02, then the Feature branch implementing a minor change shall be versioned QA03.
* **Merge:**The process of combining changes from one branch into another. In the case of using LabView, the simplest merge would be to simply replace the Main branch with the Feature branch when released.

****

## Setting up Git

### Step 1: Configure Git

Before using Git, it’s essential to configure your identity. This information will be included in your commits.

Set your username:

git config --global user.name "Your Name"

Set your email:

git config --global user.email "your.email@example.com"

You can check your configuration settings at any time using:

git config --list

### Step 2: Initialize a New Repository

Creating a New Repository

Navigate to your project directory:

cd /path/to/your/project

Initialize the repository:

git init

This command creates a new .git directory in your project folder, marking it as a Git repository.

Add files to the staging area:

git add .

This command stages all the files in your project directory for the initial commit. You can also add specific files by listing them individually.

Commit the files:

git commit -m "Initial commit"

This command commits the staged files to the repository with a message describing the commit.

Connecting Your Local Repository to the Remote Repository

Add the remote repository URL:

git remote add origin <https://github.com/AllenAircraft/mloi-ols.git>

Connect to an existing remote repository URL:

git remote set-url origin <https://github.com/AllenAircraft/mloi-ols.git>

Push your local commits to the remote repository:

git push -u origin main

### Step 3: Cloning an Existing Repository

If you want to start working on an existing project, you can clone a remote repository to your local machine.

Navigate to the desired directory where you want to clone the repository:

cd /path/to/directory

Clone the repository:

git clone https://github.com/username/repository.git

This command creates a copy of the remote repository on your local machine.

## Pull from GitHub

Typically, “pull” is used when code is being developed on two or more workstations. To download the latest code to a development station that already has a project cloned:

1. Open a command window by typing “cmd” into the Search bar.
2. Enter the drive letter followed by a colon (m:) where the project exists.
3. Change directory (cd) to the project folder.
4. Enter “git pull” to update the local copy of the project from GitHub.

A screenshot of a computer program

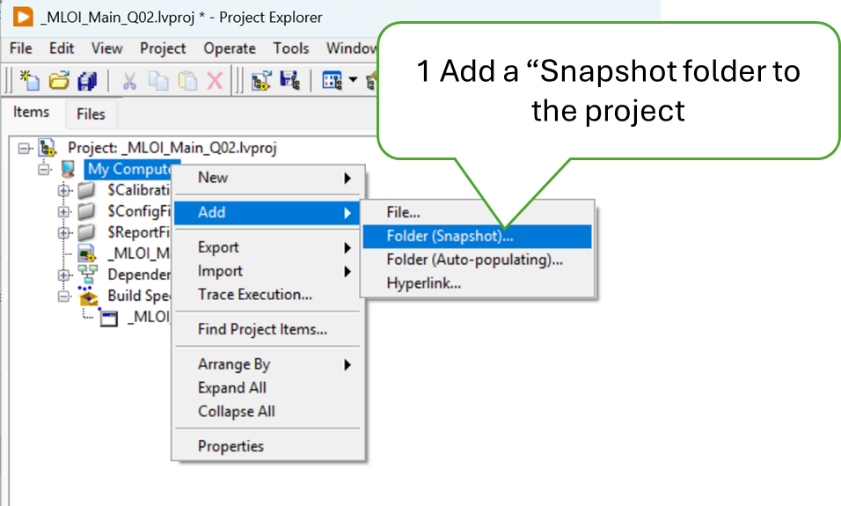
AI-generated content may be incorrect.

## Building the LabVIEW App Executable

LabVIEW sourced code can only be edited and run in the LabVIEW development system. The project configuration is maintained in the LabVIEW Project file, which also contains the system commands to build the executable. The executable can be run on any Windows PC that has LabVIEW Run-time installed.

### Configuring the LV Project

1. Add folders and files to be included to the project tree.

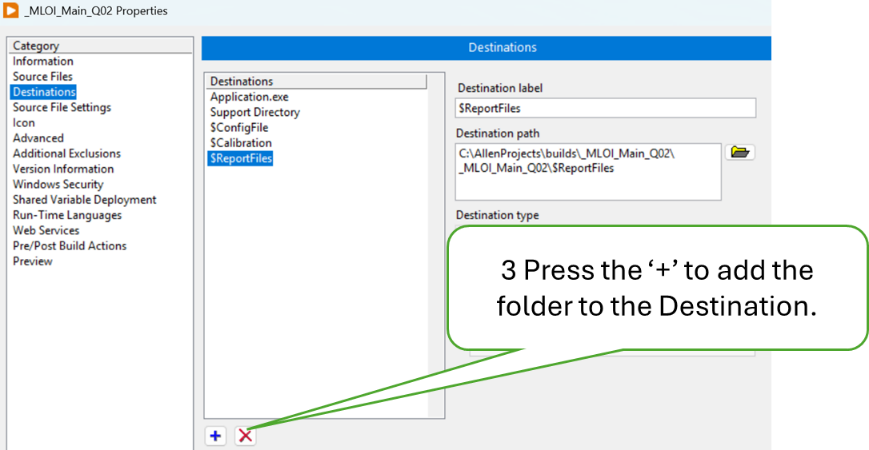


1. Include the folders and files in the project.

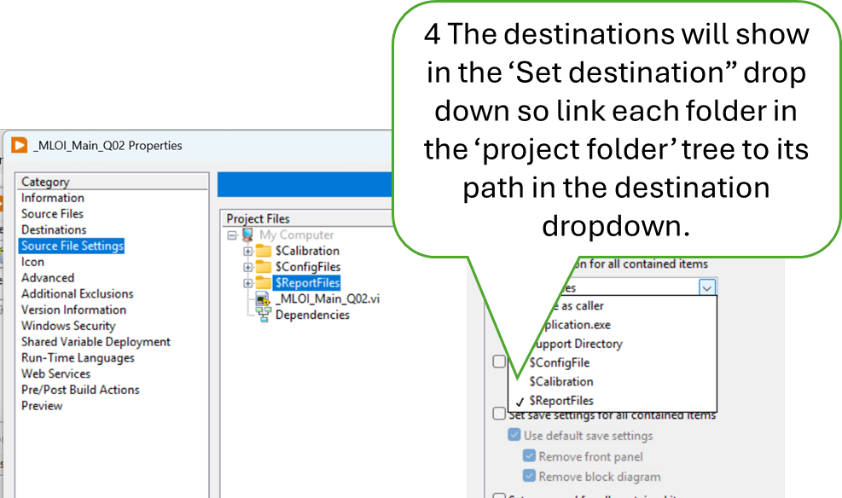
A screenshot of a computer

AI-generated content may be incorrect.

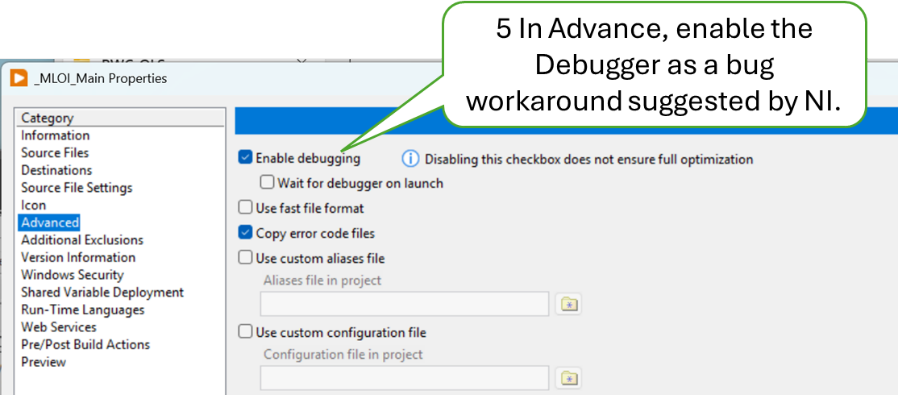
1. Add the Destination Folder for each Item to be included.



1. Link the Project Item it’s Destination Folder.

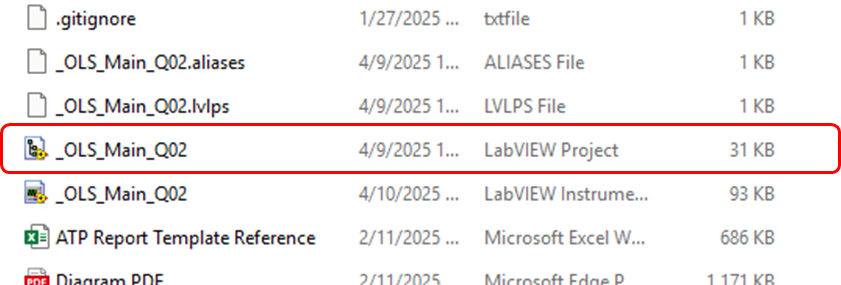


1. Turn on the debugger.



### Building the Executable

1. In Windows, navigate to the project folder, ..\PWC\_OLS
2. Select the LabVIEW Project File.

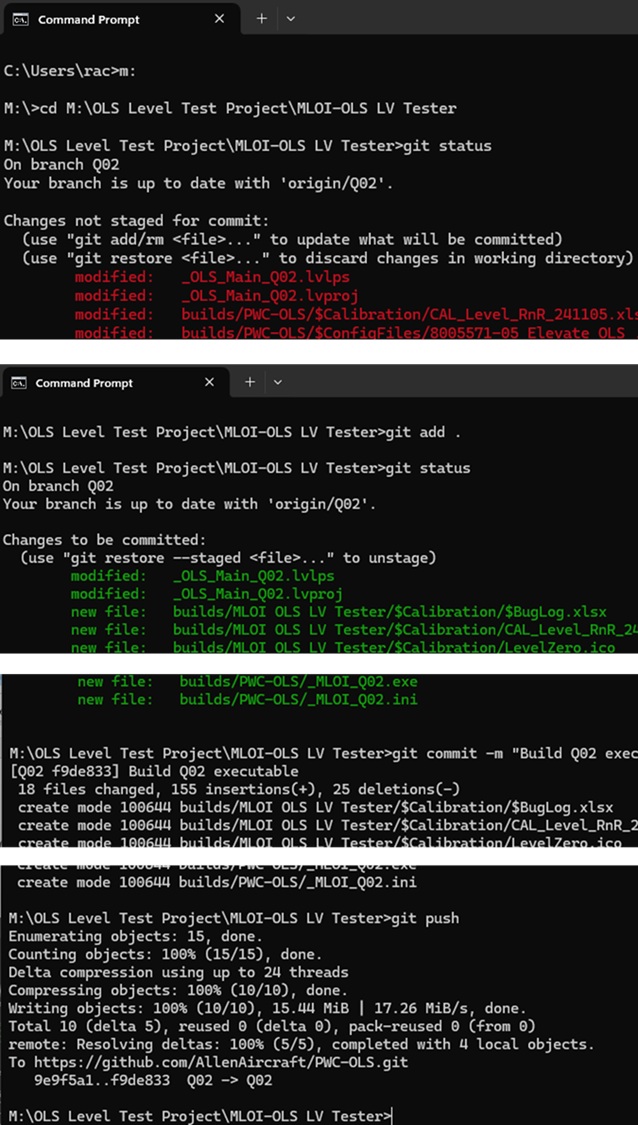
****

1. Expand the build specification branch
2. Double click on the MLOI-OLS Tester icon.

A screenshot of a computer program

AI-generated content may be incorrect.

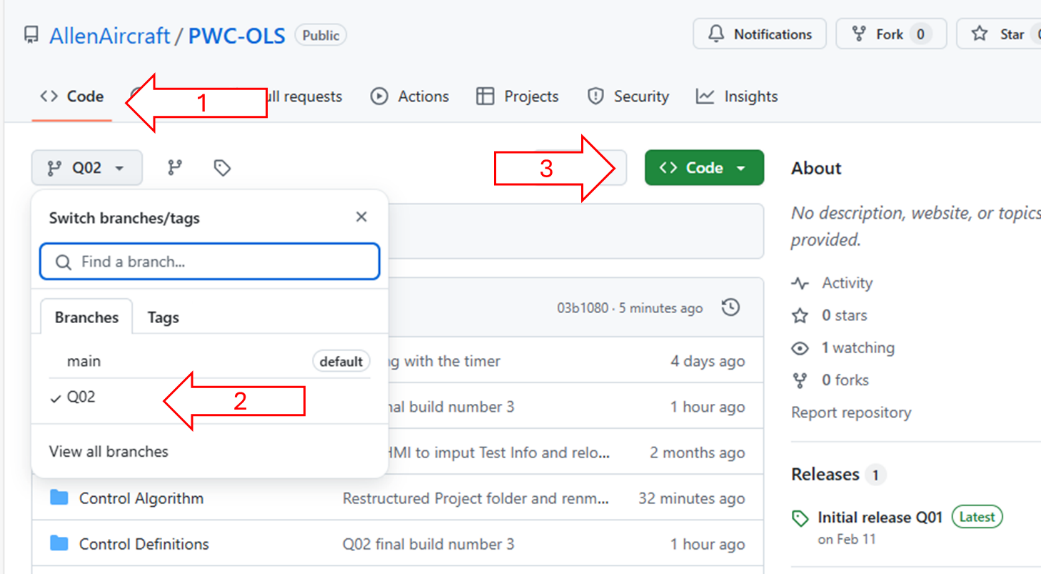
1. Click the “Build” Button.
2. The default destination folder should be named “builds” in the project folder.
3. Save and Close the project folder.
4. Open Cmd window to upload the build to GitHub
5. Navigate to the project folder
6. Then git add . (note the ‘.’ is required); git commit -m “some comment”
7. And git push does the upload to GitHub.



## Deploying the LV app to the target test-PC.

The LV app executable is saved in the Allen Aircraft project on GitHub at [GitHub - AllenAircraft/PWC-OLS](https://github.com/AllenAircraft/PWC-OLS) The easiest way to download the LV app is to download the project as a zip file and extract the MLOI-OLS LV Tester folder to the target PC desktop. Perform the following steps on the Test target PC.

1. Use an internet browser to navigate to *GitHub-Allen Aircraft/PWC-OLS*. Click on Code button.
2. Select the Version of the project to be downloaded. Note that Main is the latest Version released. Other version are either archives or under evaluation. In this example Q02 is shown, so Main is the previous version Q01. See Versioning control.



1. Click on the green CLONE button then select “Download ZIP”
2. Open the project zip file; navigate to ..\builds
3. Copy folder *MLOI-OLS LV Tester* to the Target PC, default is desktop.
4. Open the *MLOI-OLS LV Tester* folder, and create a shortcut to *\_MLOI\_Q02.exe*
5. Move the shortcut to the desktop.
6. Run the LV app by clicking on the new desktop shortcut.

## Using Git Hints

To use the Windows Power Shell to run Git

1. Use Windows explorer to navigate to the project folder where the ..git file resides.
2. Right click in the white space, pick “Open in Termimal,;

A $ indicates input at the system prompt. The ellipses (. . .) separate the input from the response.

### Creating a Local Project Folder

Navigate to the folder where you want the project folder to be created.

$ m:

$ cd “M:\OLS Level Test Project”

Browse to GitHub Repository, and copy the link under green Code button. Clone the project by pasting the copied link into the clone command.

$git clone <https://github.com/AllenAircraft/MLOI-OLS.git>

. . .

Cloning into 'MLOI-OLS'...

remote: Enumerating objects: 816, done.

remote: Total 816 (delta 0), reused 0 (delta 0), pack-reused 816 (from 1)

Receiving objects: 100% (816/816), 154.99 MiB | 50.92 MiB/s, done.

Resolving deltas: 100% (396/396), done.

Updating files: 100% (580/580), done.

Change directory to the project folder

M:\OLS Level Test Project>cd MLOI-OLS

, , ,

M:\OLS Level Test Project\MLOI-OLS>

Use checkout to select the working branch for this session.

$ git checkout QA02

. . .

Updating files: 100% (269/269), done.

branch 'QA02' set up to track 'origin/QA02'.

Switched to a new branch 'QA02'

### Switching to a Remote Branch

To view origin link use Remote command

$ git remote -v

. . .

origin https://github.com/AllenAircraft/MLOI-OLS.git (fetch)

origin https://github.com/AllenAircraft/MLOI-OLS.git (push)

To see the current local branch

$ git branch

...

\* main

q03

Start by fetching from the remote repository to make sure you have all the latest changes downloaded. This will fetch all of the remote branches.

$ git fetch -v

...

From https://github.com/AllenAircraft/MLOI-OLS

\* [new branch] Q03 -> origin/Q03

\* [new branch] Q04 -> origin/Q04

To see the branches available for checkout with:

$ git branch -v -a

...

\* Q02 8077ba7 Q02 build 10030

remotes/origin/HEAD -> origin/Q02

remotes/origin/Q02 8077ba7 Q02 build 10030

remotes/origin/master 47df981 Change Name, old Q02

The branches that start with remotes/\* can be thought of as read only copies of the remote branches. To work on a branch you need to create a local branch from it. This is done with the Git command switch giving it the name of the remote branch

$ git switch Q03

. . .

Updating files: 100% (233/233), done.

branch 'Q03' set up to track 'origin/Q03'.

Switched to a new branch 'Q03'

### Switching to a Previous Commit

Find the commit of interest.

$ git log

, , ,

commit 6bd62e3a108793d552bd9fc1bac554126532c66e (HEAD -> QA02, origin/QA02)

Author: Rick Ales <ricka@allenaircraft.com>

Date: Thu Apr 17 09:49:13 2025 -0400

Q03 build 10040 mass compiled, built and runs no errors, still needs new Motor controller

commit 387ad1f989b2ec2ba121d89065491e0907f49afd

Author: Rick Ales <ricka@allenaircraft.com>

Date: Thu Apr 17 09:20:52 2025 -0400

Removed all vi within Allen Projects folder, only mXXX.vi and defs remain

commit 4e630795c203fbff75d286de08a1b85b0fb694ad

Author: Ri. . .

Switching to a different commit, detached the pointer from the head as it now points to the requested commit. There are two ways to switch back to the head

1. Long way

Copy the commit ID (highlighted yellow above) and save to notepad file. This is needed to reset the pointer to HEAD. Copy the commit ID (highlighted cyan above) into the checkout command.

$ git checkout 387ad1f989b2ec2ba121d89065491e0907f49afd

1. Short way

$ git switch –

If either way results in an error then you must either stash the change or undo your changes with the restore (highlighted in cyan):

$ git switch -

error: Your local changes to the following files would be overwritten by checkout:

\_OLS\_Main\_Q02.vi

builds/PWC-OLS/$Calibration/CAL\_Level\_RnR\_241105.xlsx

builds/PWC-OLS/$ConfigFiles/8005571-05 Elevate OLS .csv

builds/PWC-OLS/\_OLS\_Main\_Q02.exe

Please commit your changes or stash them before you switch branches.

Aborting

$ git branch

\* (HEAD detached at 9e9f5a1)

Q2\_01

main

$ git restore \*

$ git switch -

Previous HEAD position was 9e9f5a1 fix test protocol bug, redesigned timer, added some tool tips

Switched to branch 'Q2\_01'

Your branch is up to date with 'origin/Q2\_01'.

$ git branch

\* Q2\_01

main

### Create a new branch and save to remote GitHub

Starting fron an exixting branch

$ git status

\* On branch Q2\_01

Your branch is up to date with 'origin/Q2\_01'.

Create a new local branch

$ git checkout -b Q2\_02

\* Switched to a new branch 'Q2\_0

$ git status

\* On branch Q2\_02

nothing to commit, working tree clean

PS C:\AllenProjects\PWC-OLS> git push Q2\_02

.

$ git push -u origin Q2\_02

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)

remote:

remote: Create a pull request for 'Q2\_02' on GitHub by visiting:

remote: <https://github.com/AllenAircraft/PWC->OLS/pull/new/Q2\_02

remote: To https://github.com/AllenAircraft/PWC-OLS.git

\* [new branch] Q2\_02 -> Q2\_02

branch 'Q2\_02' set up to track 'origin/Q2\_02'.

To switch to the new branch

$ git branch

\* Q2\_01

Q2\_02

Main

$ git switch Q2\_02

\* Switched to branch 'Q2\_02'

Your branch is up to date with 'origin/Q2\_02'.

### Add a Prefix to the Names of Multiple Files in Power Shell

Change directory to where the files are contained

Copy and Paste the following command into power shell, then change the “x” to what ever text to want to prepend to all the file names in the directory.

$ cd test

$ (Get-ChildItem -File) | Rename-Item -NewName {$\_.Name -replace "^","x"}

$ dir

Directory: C:\AllenProjects\PWC-OLS\test

Mode LastWriteTime Length Name

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

-a---- 4/26/2025 6:35 AM 28306 x$CTL\_LevelCtlr.vi

-a---- 4/26/2025 6:35 AM 30261 x$CTL\_LevelCtlrQ02.vi

-a---- 4/26/2025 6:35 AM 17868 x$CTL\_Limit Check.vi

(Get-ChildItem -File) | Rename-Item -NewName {$\_.Name -replace "^","x"}

A screenshot of a computer

AI-generated content may be incorrect.