Automated Wireless Research Tester
User Manual

User Guide

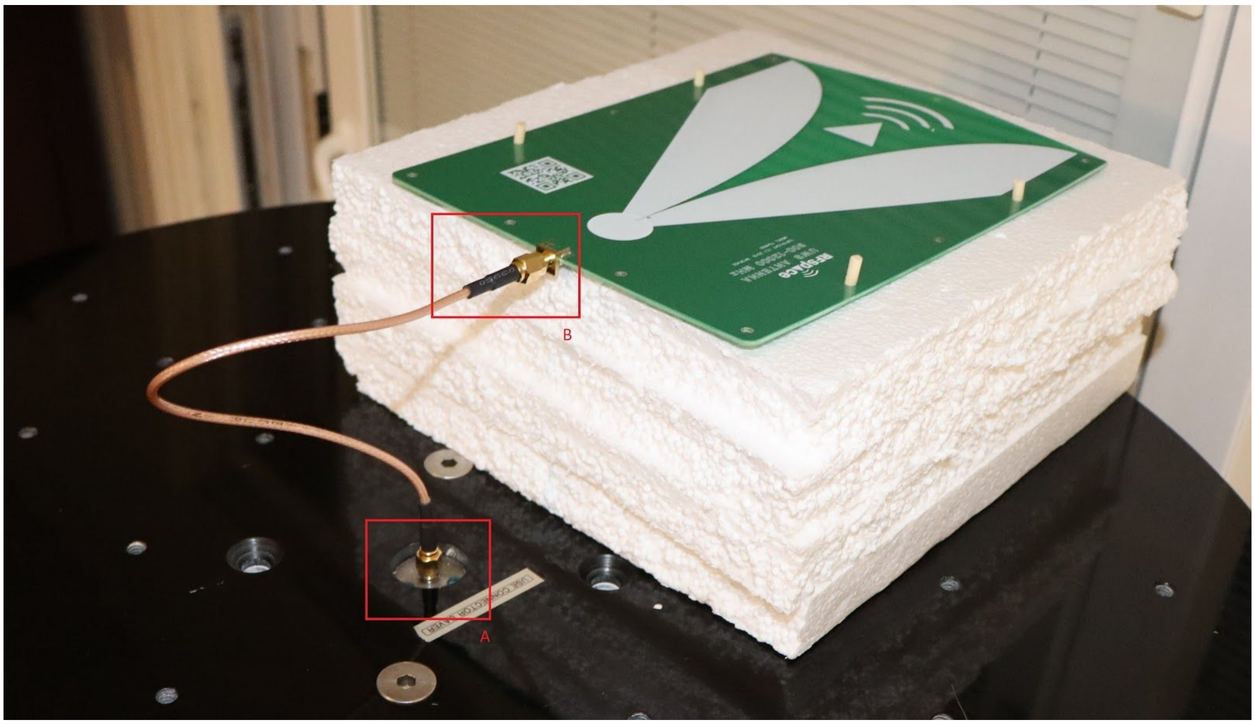
The first portion of this document is for users who are interested in using the software. In the following sections, we will go through how to set up your hardware and software before using the software, and then we will go through a tutorial for the software itself.

Hardware Setup

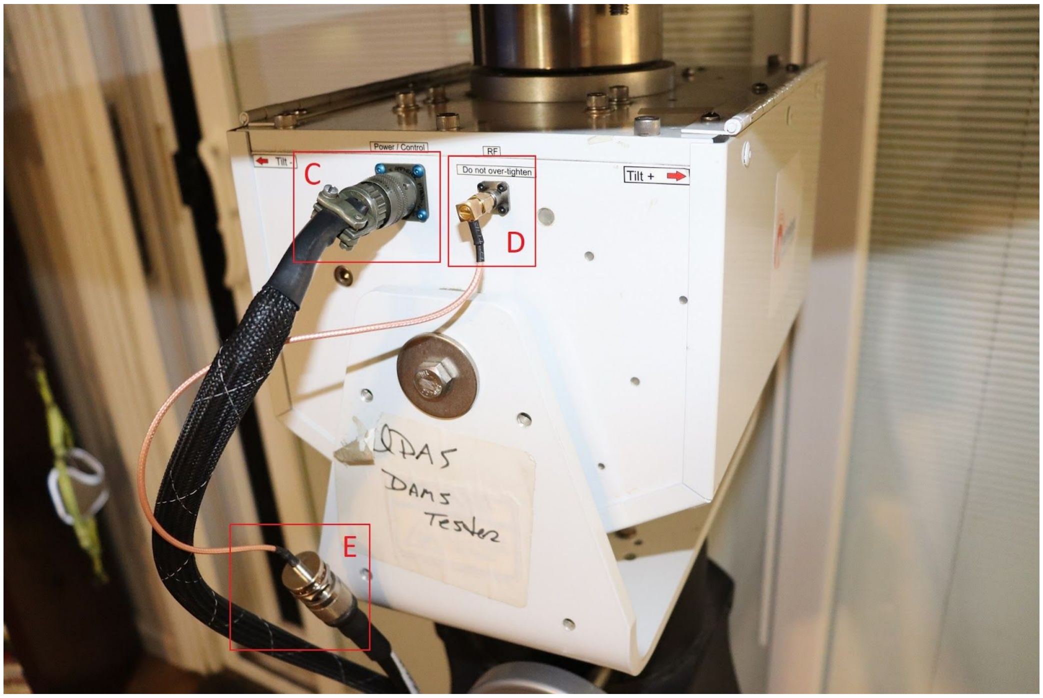
Positioner, Antenna Under Test, and Reference Antenna

The positioner and antenna under test have three components to connect when setting up the test system, the power supply/communication line to the positioner, the RF cable to the positioner, and an RF connector wire between the slip-ring in the middle of the mounting plate on top of the positioner, and the antenna being tested. Additionally, mounting the antenna on the mounting plate is required, but the correct mounting mechanism is left to the user to determine; here a styrofoam block was used to place the antenna on, with wooden dowel rods to hold the block in place. The reference antenna will have an RF connection that must be made which will be the same as the one to the antenna under test. Mounting of the reference antenna is left to the user of the system to determine.

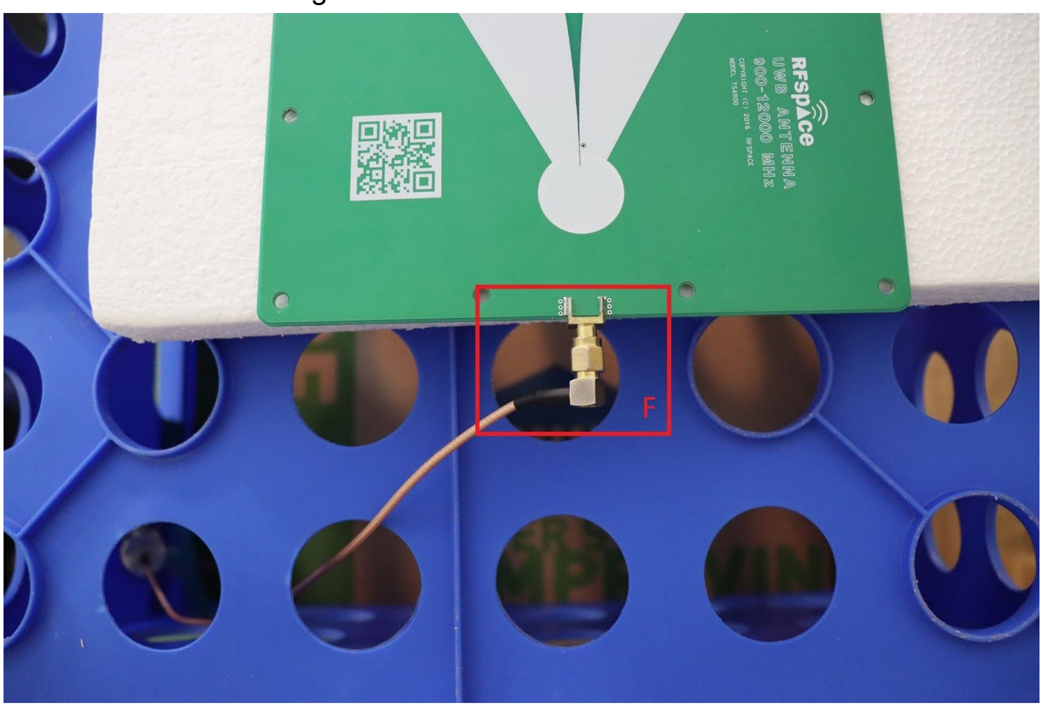
The connection of the RF connector wire to the slip-ring on the positioner is shown in box A, and the RF connector wire to the antenna under test in box B.



The connection of the positioner power supply/communication wire is shown in box C, and the RF cable connections are shown in boxes D and E.



The connection of the RF cabling to the reference antenna is shown in box F.

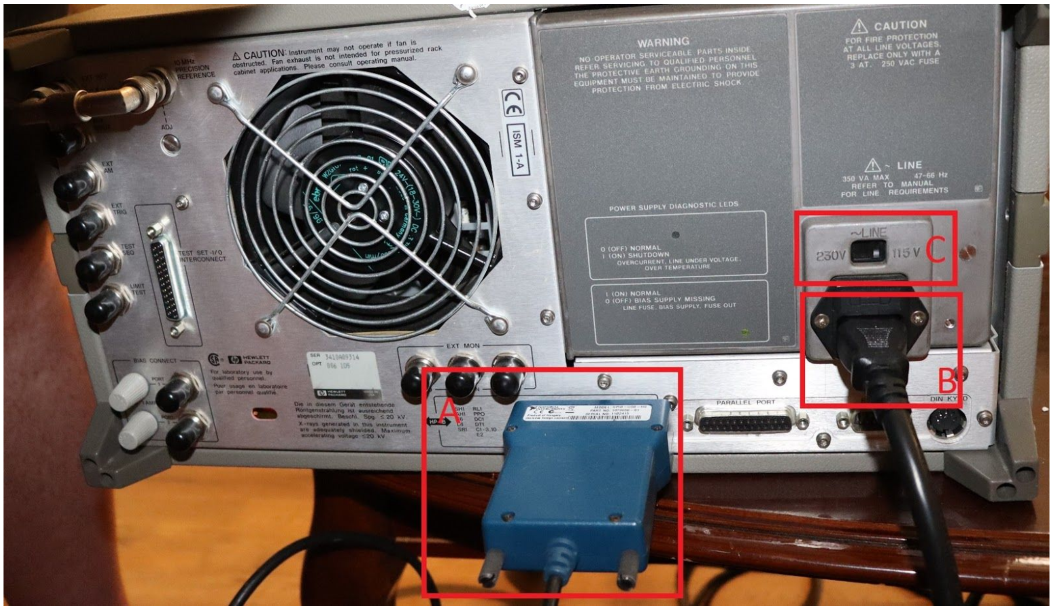


The termination of the other ends for the positioner power supply/communications cable and RF cabling will be shown in the Laptop and Vector Network Analyzer sections respectively.

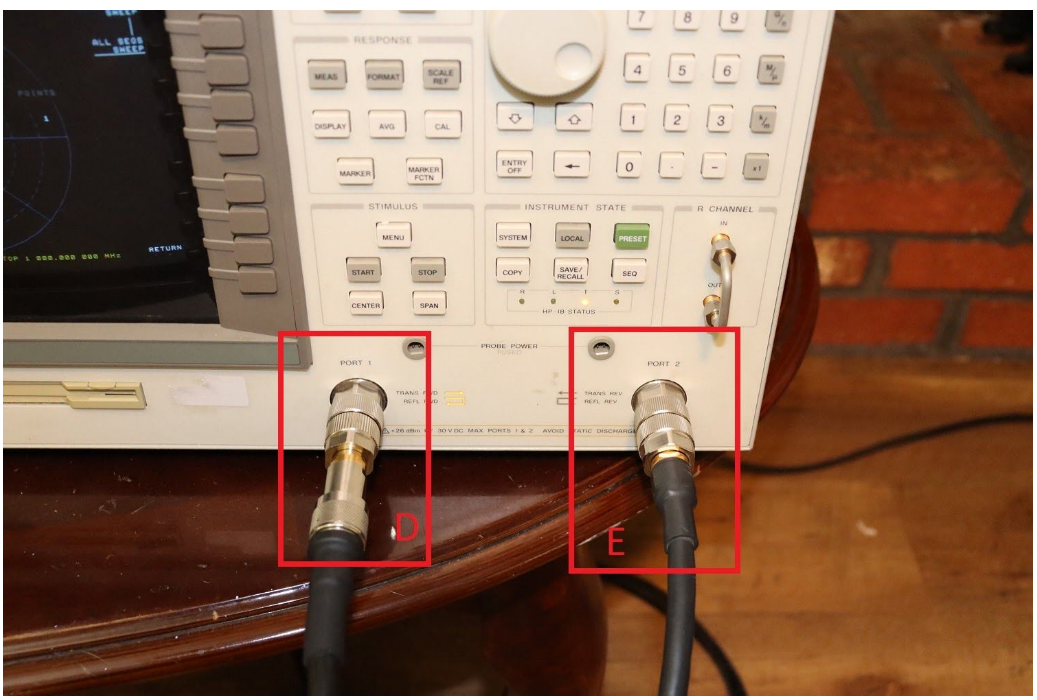
Vector Network Analyzer

The Vector Network Analyzer (VNA) has several connections which must be made: the two RF cables which are connected to the two antennas being used, the GPIB adapter which connects the instrument to the laptop, and a power supply

The GPIB and power supply connections are relatively straightforward. The power supply is from a 120V source and is a cable similar to those used with desktop computers - make sure the voltage selection switch is correctly set to 115V, and the GPIB connection is a 32 pin port. Box A shows the GPIB connection, box B shows the power supply connection, and box C shows the voltage selection switch.



The connection of the RF cables for the antennas being used is slightly more complex due to the type of connector. First of all, the connection between the antenna under test is made to Port 1, shown in box D, and the reference antenna is connected to Port 2, shown in box E.



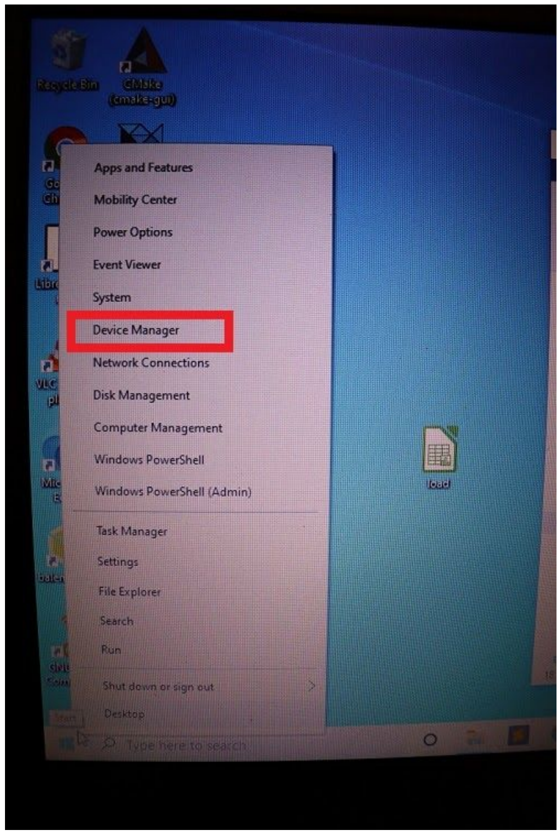
To make these connections, first back off the external portion of the port on the VNA as shown below

Next, screw the external portion of the adapter down onto the exposed threaded section of the port on the VNA. Note: DO NOT tighten these connections on both sides, only tighten one.

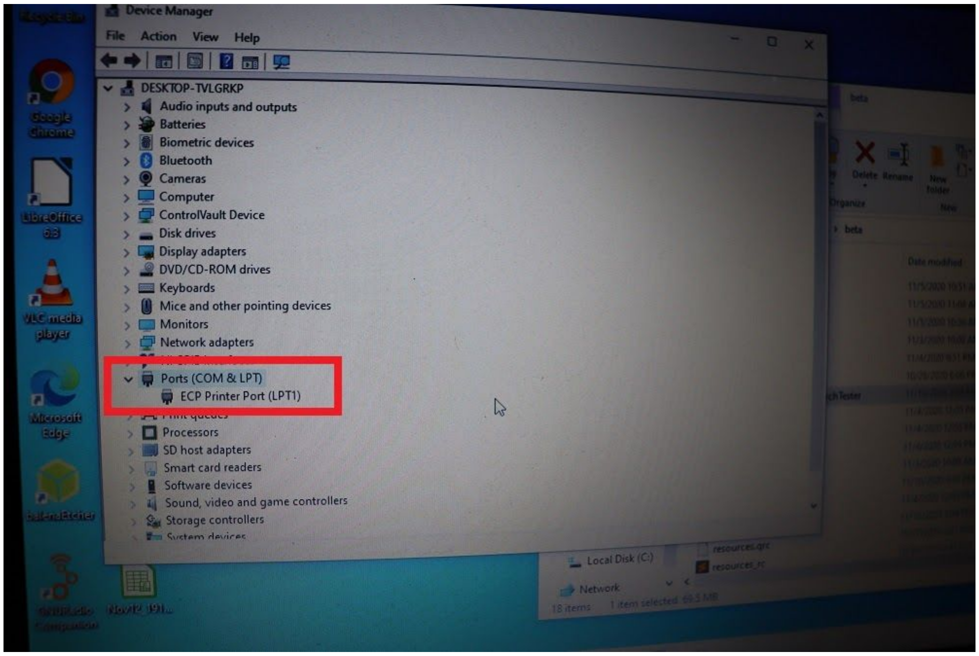
Laptop

The laptop setup is relatively straightforward, it has a power supply, and the only additional connections are the ones made from the positioner communications cable and the VNA GPIB adapter. These are both USB connectors, so if the laptop does not have more than one USB port a hub will be needed.

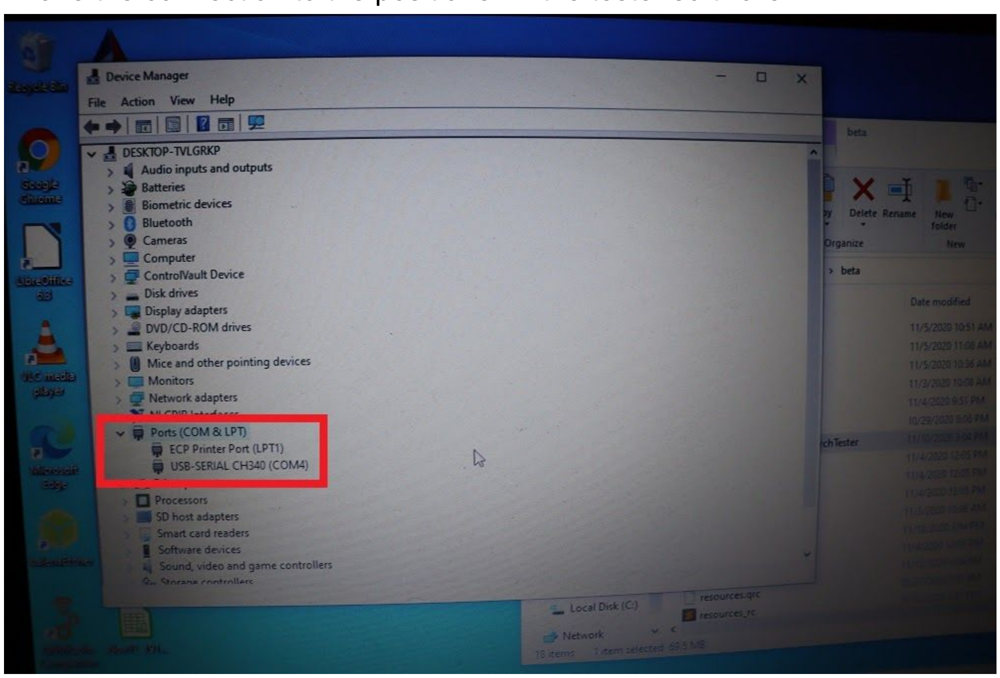
One piece of information that will need to be determined on setup is the COM port used by the positioner. To determine this, first open the windows device manager.



Next, find the Ports (COM & LPT) header and expand it to see what is currently listed.



Finally, plug in the positioner and recheck the Ports section in the device manager, the added entry will be the COM port assigned to be used by the positioner connection; as shown below in this example the positioner is assigned to the COM4 port. Additionally, right click on the port, select *Properties* from the menu, click on the *Port Settings* tab, and find the baud rate (bits per second) the port is set to. Write down the port number and baud rate, this information will be needed to make the connection to the positioner in the tester software.



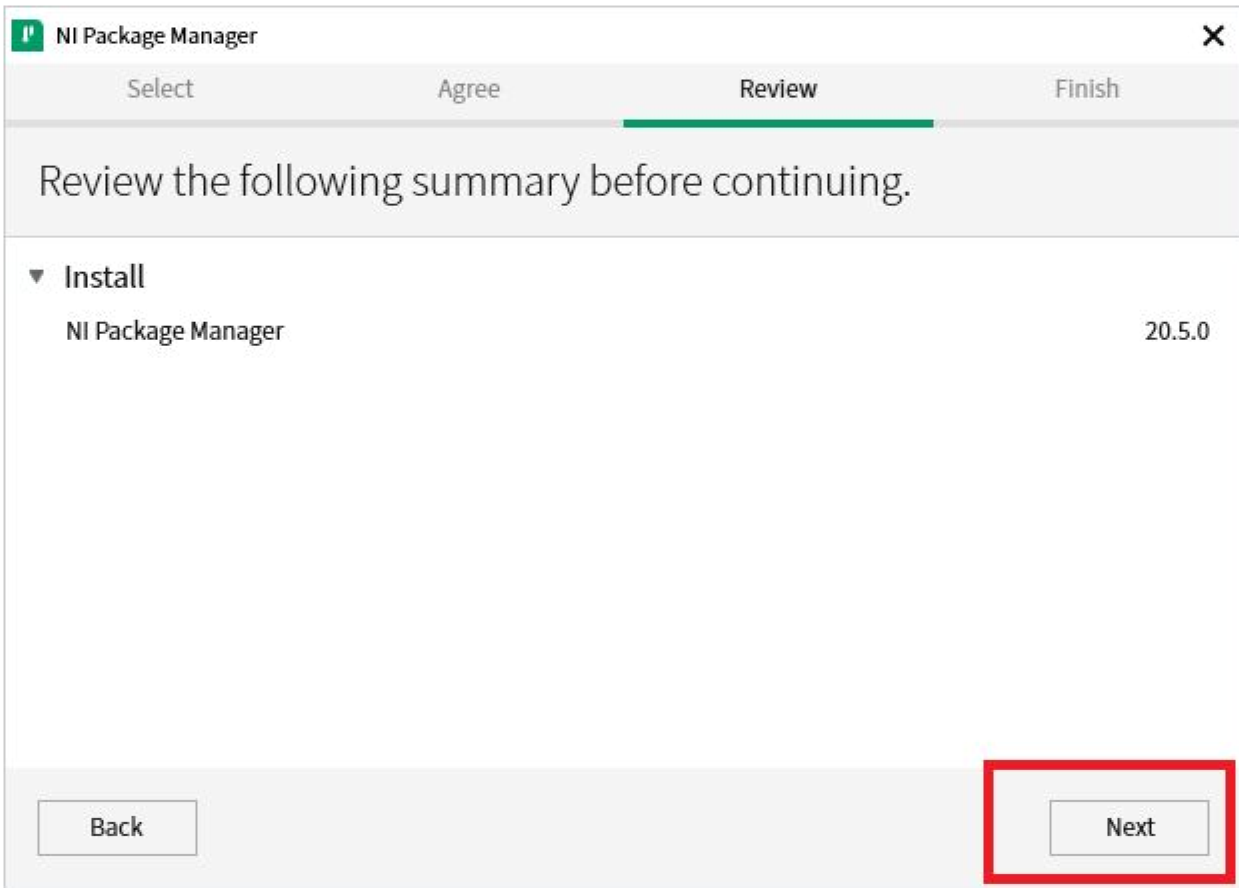
Software Installation

Installing Drivers

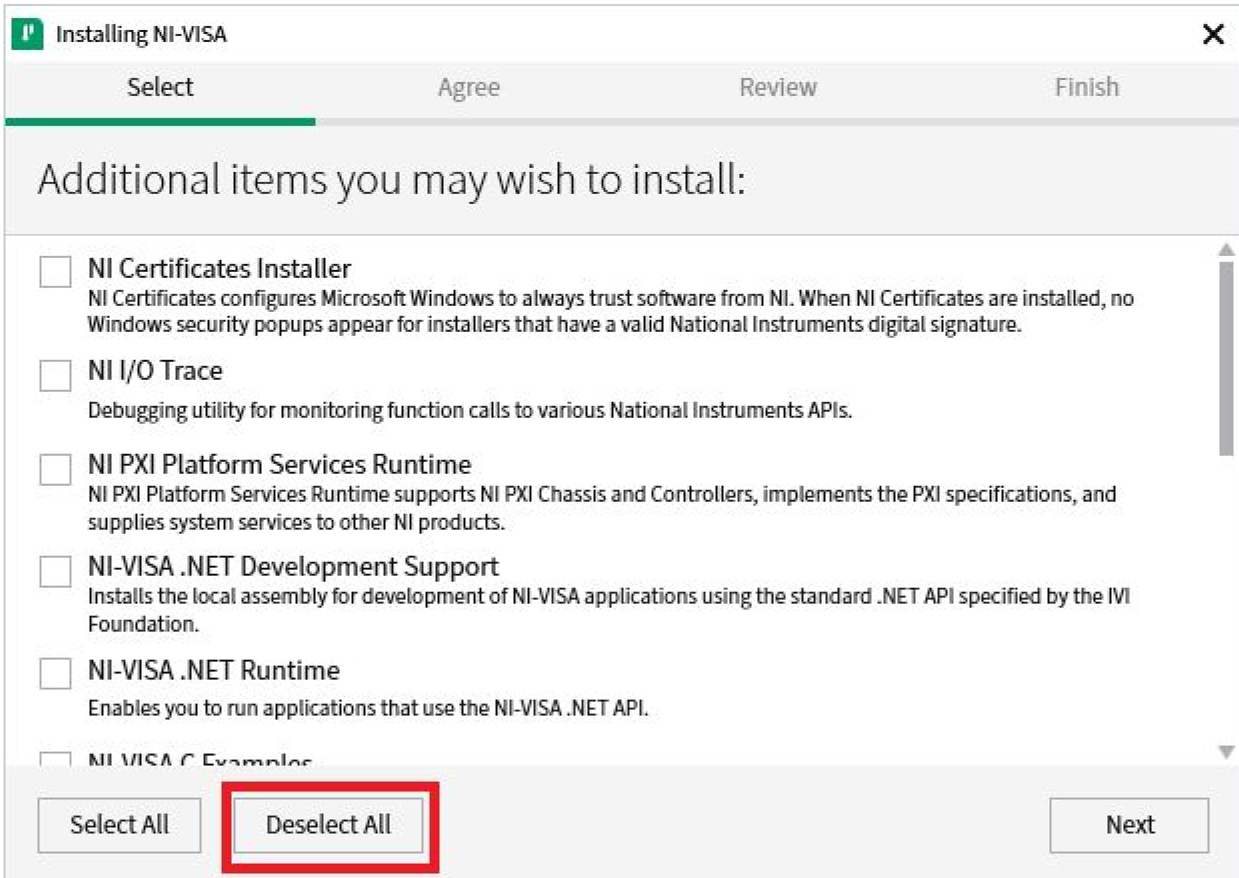
This software uses drivers from National Instruments in order to control both the VNA and the positioner. Please go to the following links and download the drivers:

- NI-VISA Download (Version 19.5 or newer): https://www.ni.com/en-us/support/downloads/drivers/download.ni-visa.html
- NI-488.2 Download (Version 19.5 or newer): https://www.ni.com/en-us/support/downloads/drivers/download.ni-488-2.html

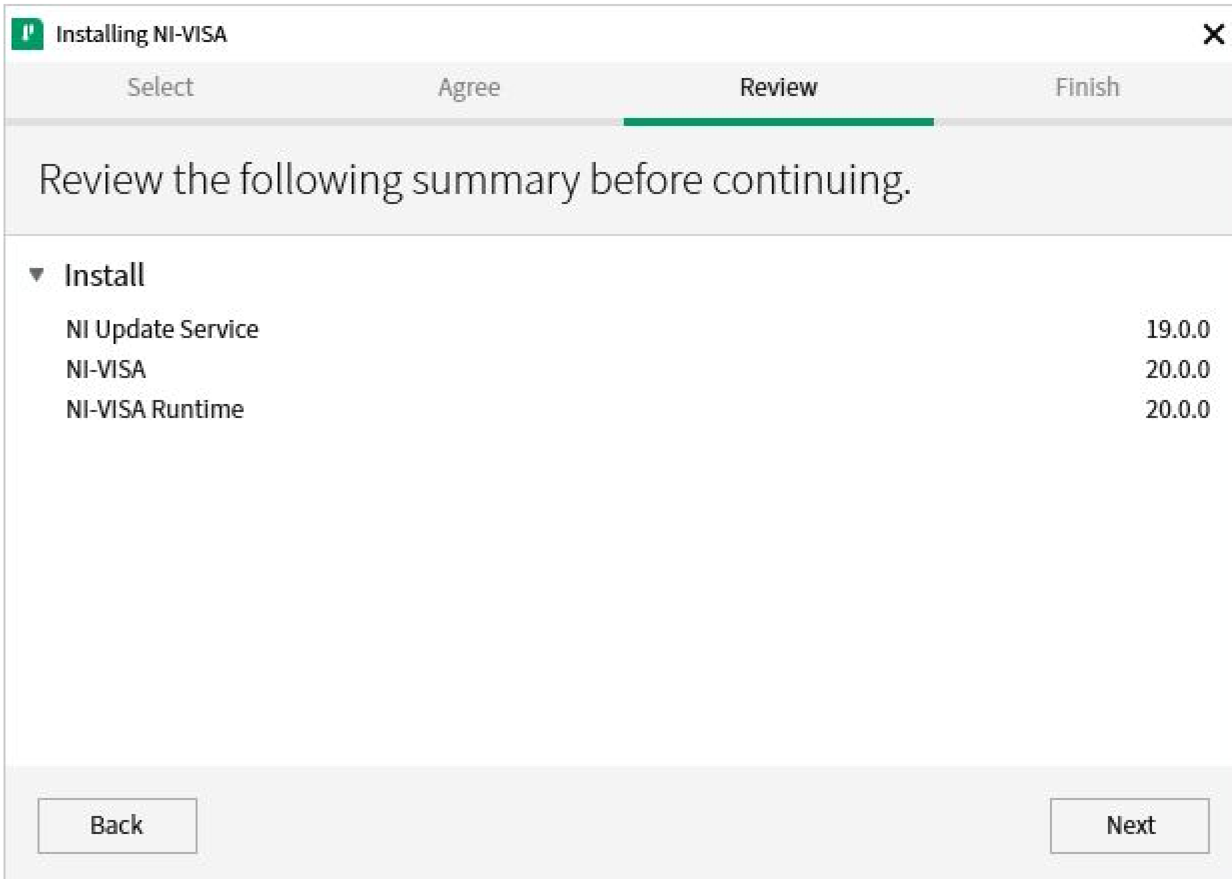
The installer will prompt you to install the Package Manager first, click Next to continue:



Next, the installer will give you the option to install additional items. These items are not required for our software. You may click “Deselect All” and continue:



Next, the installer will list all the items that will be installed. It should look something like this:

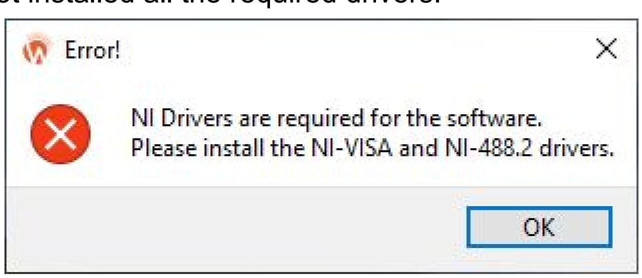


Continue on and finish installing. The process will be the same as above for the NI-488.2 Driver.

Starting the Software

To start the software, locate `AutomatedWirelessResearchTester.exe` in the software folder. Simply double click on the file to run it. Feel free to add a shortcut to your desktop for quicker access.

Note: If you encounter the error message below when attempting to start the software, that means that you have not installed all the required drivers.



Using the Software

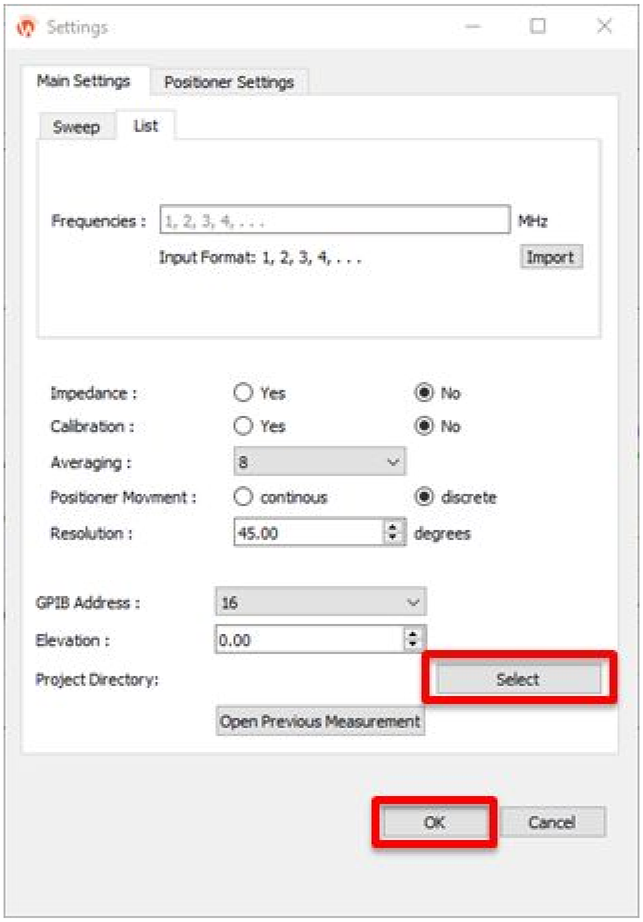
Once the window has been opened, it should look like the figure below:

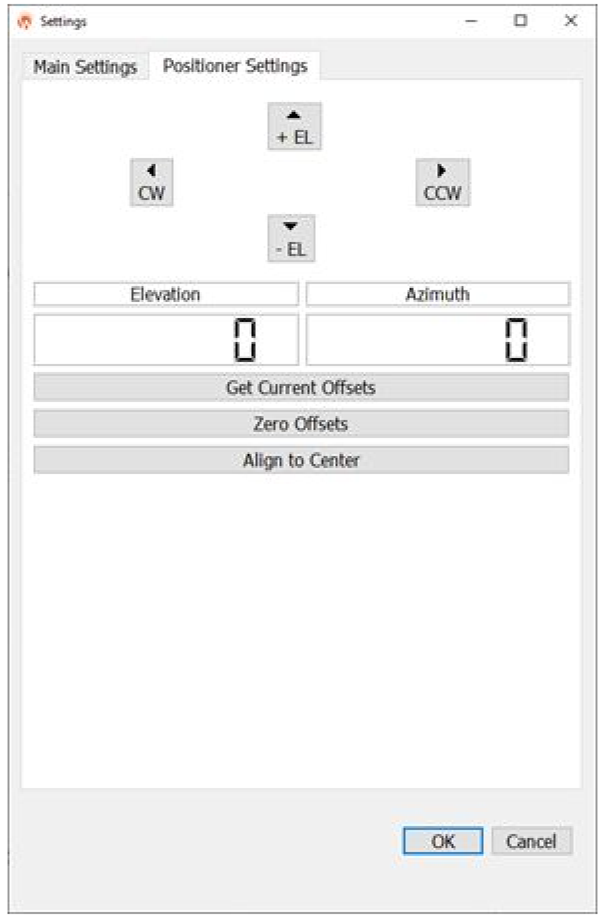
Note: If you are experiencing some difficulties with the display of the graphical user interface, change your computer's display scale to 100% and reopen the window to update the changes.

Setting boundary parameters

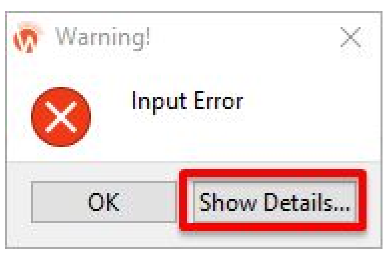
To begin the antenna measurement, you have to set the parameters for the positioner and the network analyzer.

1. Please click the Settings Icon from the Main Window to open the Settings Window.
2. The Settings window will have two tabs as seen below: The Main Settings (left) for the Network Analyzer input and the Positioner Settings (right) for the Positioner input.
3. Fill in and choose the necessary parameters needed
4. Select a file location to save your current project and click OK.



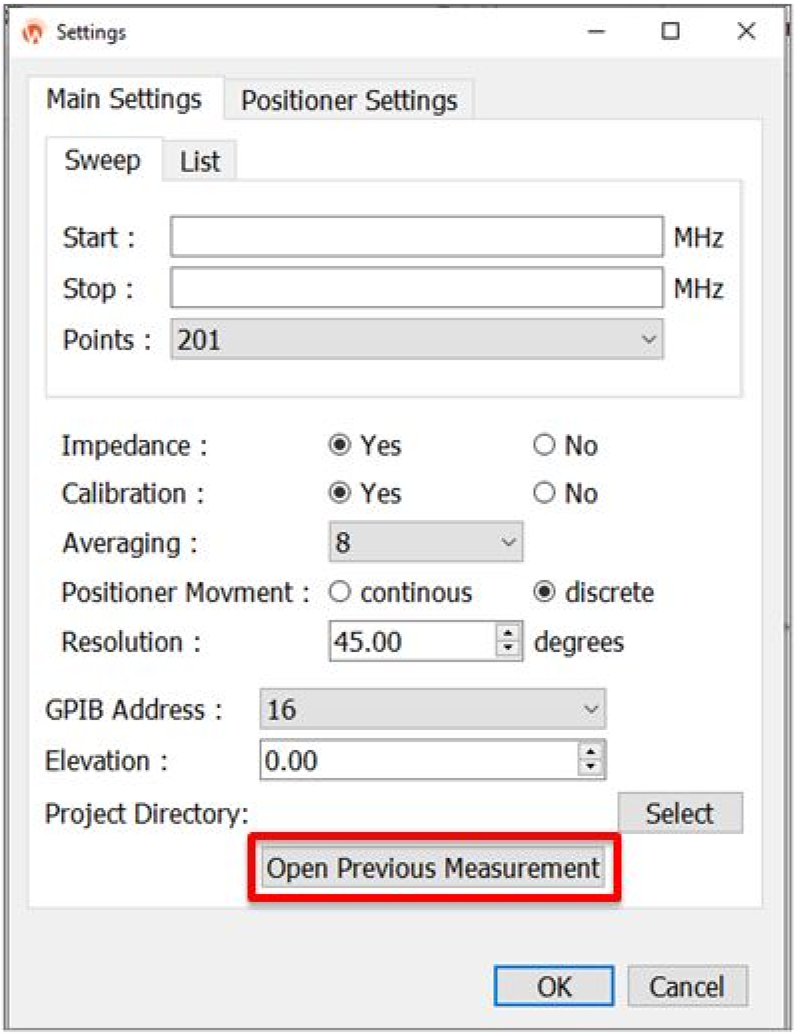


Note: If you receive an error message, that means you have failed to comply with the input restrictions. The Show Details button, as seen below, will help you find the error in your input.



Opening existing data

From the Settings Window, click the Open Previous Measurement button and pick the .csv file you want to view. Once you have picked the file, the system would then redirect you to the Graphing Window to view your chosen file.

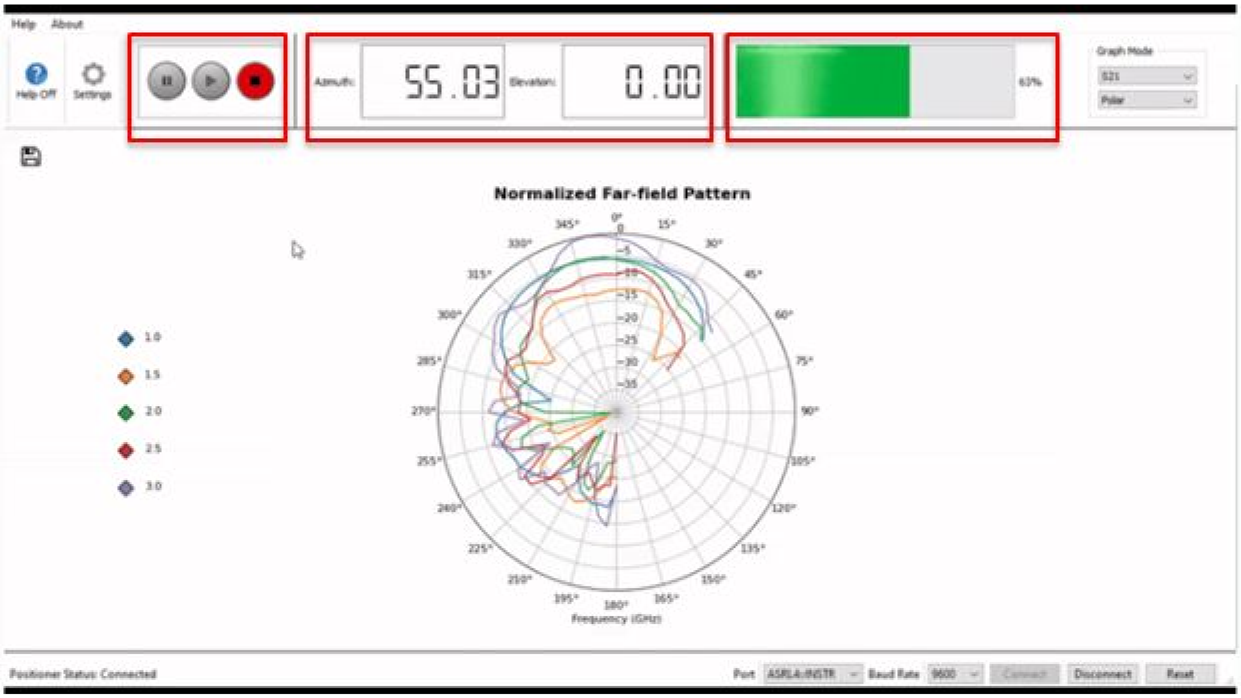


Exploring graphical window

The graphing window plots the normalized far-field patterns taken from the antenna under test.

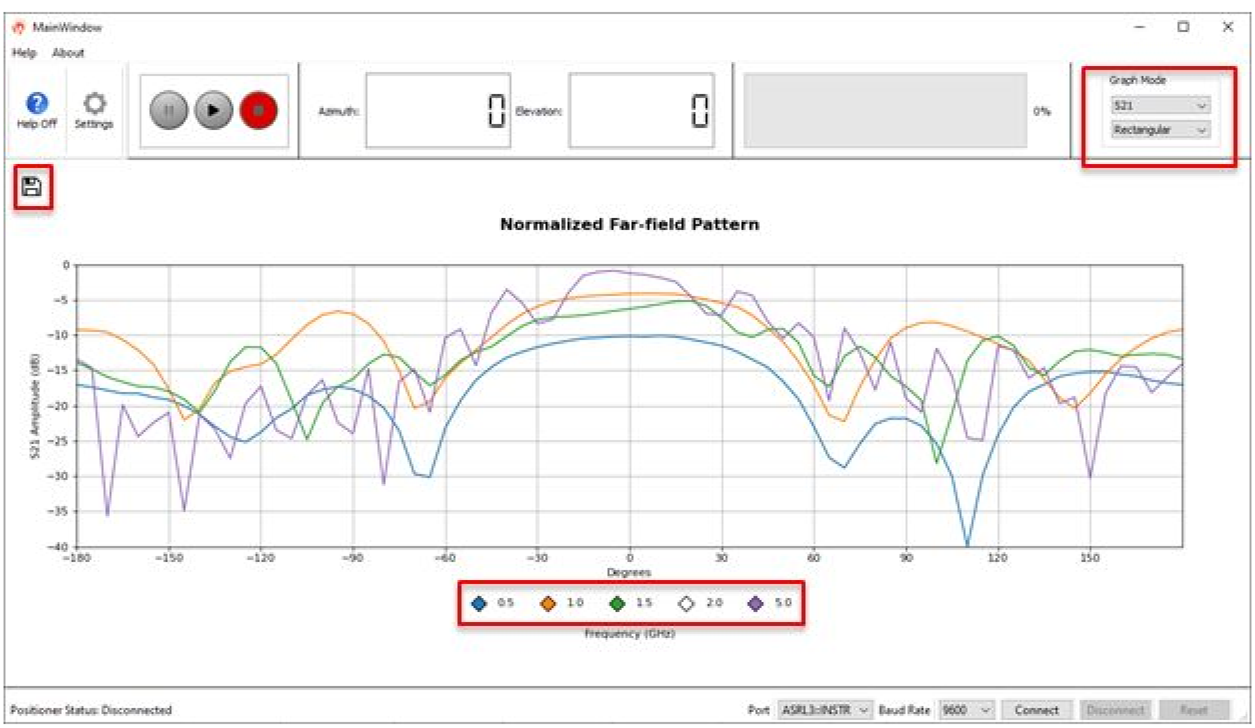
Once the settings parameters have been accepted, the graphing window would plot the pattern in real time, show the progress time, and the Azimuth and Elevation if there are any.

An additional option to pause, play, or stop measurements have been implemented as an equipment safety precaution.



After the measurement has been finalized you have the option to do the following options:

- Change the graph mode in either Impedance (rectangular form) or S11 (polar or rectangular form).
- Select or deselect frequency values by toggling the diamond icons next to the plot.
- Save the figure in jpeg format.



Checking settings input restrictions

The software is programmed to have error checking functions to prevent incorrect values to be passed on to the equipment and the data processing block. Listed below are some of the restrictions:

a. Users can either enter a linear or list frequency but not both.
b. Input should be either an integer or float number.
c. The list of frequencies should be entered according to the arranged format.
d. Values for the list frequency should be within the range of [0.03, 6000] MHz.
e. Impedance requires calibration, selecting impedance automatically selects calibration.

An error message with a short explanation will help the user resolve the problem and once the error has been resolved, the software will proceed with the antenna measurements.

Getting additional help

- To receive additional information about the parameters within the software, simply follow either of these steps:
  - Click the Help Icon on the top left of the window and hover over the parameter.
  - From the toolbar click Help > Turn Help on and hover over the parameter.
  - To disable, you can either click the Help taskbar or Help icon to toggle off information.

- To access the hard copy documentation, click Help > Documentation from the window toolbar.

Software Programming Guide

This remaining contents of the document will cover everything you need to know to modify the source code of the software. This will include the list of tools you will need for programming, a high-level overview of the main modules of the code, and finally how to repackage your source code into software.

Required Tools

The following table contains all of the resources (all free) you would need to modify and test your source code:
Note: *It is recommended that you create a virtual environment like anaconda to install these Python packages so that the different versions do not interfere with your local python environment.*

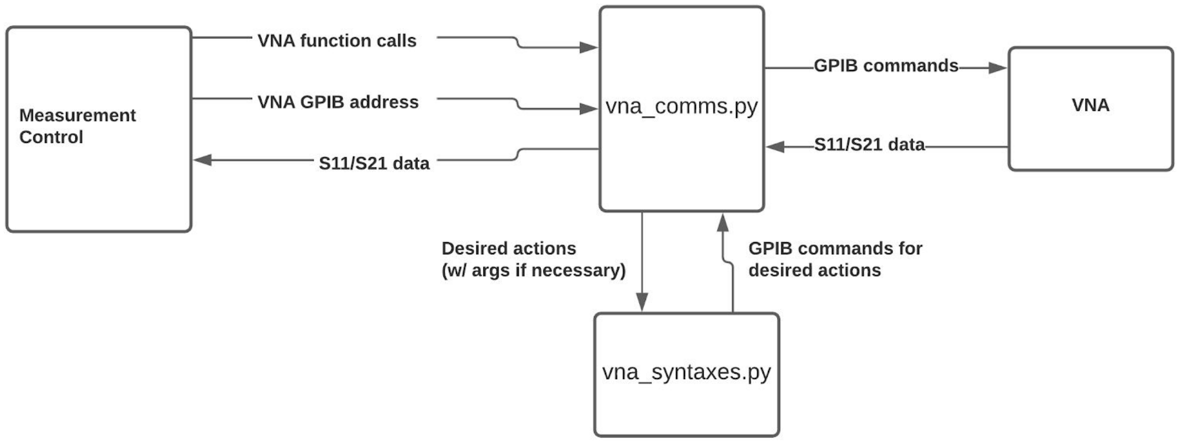
Name	How it’s used	How to install
Python (3.8 recommended) (*Install this first!*)	It is the main programming language in the source code.	Visit Python to download
PyQt5 (5.15.1)	Python package used for programming our graphical user interface.	Type `pip install PyQt5==5.15.1` in the command prompt
matplotlib (3.2.2)	Python package used for generating plots.	Type `pip install matplotlib==3.2.2` in the command prompt
pandas (1.1.4) and numpy (1.19.3)	Python packages used for reading and processing data files.	Type `pip install pandas==1.1.4 numpy==1.19.3` in the command prompt
PyVISA (1.11.1)	Python package used for communicating with the positioner and VNA.	Type `pip install pyvisa==1.11.1` in the command prompt
Qt Designer	Tool to design the layout of the graphical user interface.	Visit Qt Designer to download
pyinstaller (4.0)	Tool to package python source code into an executable	Type `pip install pyinstaller==4.0` in the command prompt
National Instruments Drivers	Required drivers to control the positioner and VNA	See the Installing Drivers section

Measurement Control

Measurement Control - Positioner

Measurement Control - Vector Network Analyzer (VNA)

The VNA portion of measurement control is responsible for communicating data to and from the VNA. The figure below shows how measurement control interacts with the VNA modules:



In `vna_comms.py`, all of the VNA functions are located inside the `Session` class. These functions include:

- `reset_all`: Performs a factory reset on the VNA
- `reset`: Resets only measurement parameters on the VNA
- `setup`: Configures the VNA with the appropriate sweep/frequency list, averaging factor, and IF bandwidth.
- `get_data`: Will take 1 data point (either S21 or S11) at the frequencies indicated in `setup()`. The function returns a list of data points, with each data point being a Data class object, which contains information about measurement type, frequency, position, and value of data.
- `calibrate`: There are three separate calibration functions: One for each of standards that need to be calibrated (open, short, and load).
- `rst_avg`: Resets the averaging on the VNA. This is usually used whenever the positioner moves to the new position, and we don't want the averaging to contain data from previous positions.
- The `Session` class also maintains the communication session with the VNA (think of it like being on a phone call). Hence the name session.

Basically, measurement control creates a `Session` class object. From there, measurement control will call specific VNA functions depending on where we are in the measurement. The VNA function calls will then turn into GPIB commands, which will then be passed on to the VNA itself.

This is where `vna_syntaxes.py` comes in. `vna_syntaxes.py` contains a number of functions, with each function representing an action to be performed on the VNA. The functions take accept arguments including the VNA model and the command arguments (e.g. frequency), and it returns a string containing the properly formatted GPIB command for this specific action(s) on the specified VNA model.

Adding a New VNA

The structure in `vna_syntaxes.py` is set up in such a way that allows you to add additional VNAs that would be able to be used with the software. We will go through a short example to show you all the pieces you will need to add to `vna_syntaxes.py`:

1. Adding the model information: The first thing you will need to do is add your new VNA to the model class. See the screenshot below for an example:

```
class Model(Enum):
    """Add additional VNAs here"""
    HP_8753D = auto()
    # ex: NEW_VNA = auto()

```

2. Identify the model: Now that you've added the model, there needs to be a way for the code to know which VNA model it is communicating with. An ***IDN query** (identification query command) is sent to the VNA, and the VNA a string with the model information. **To obtain the returned string, try running the script in `vna_identify.py`**. It will ask you the GPIB address of your VNA, and it will print the returned identification string.
   Once you have obtained the string, add an else if statement to the block below in check_model:

```
def check_model(string):
    """Indicate a unique portion of the returned *IDN? query string
    which can identify the specific VNA model"""
    if '8753D' in string:
        return Model.HP_8753D
    else:
        raise Exception('Model is either not supported, or model is not
```

3. Adding all the commands: Now comes the tedious part. You will likely need to reference the programming guide for VNA so that you have information on all the different GPIB commands that you can send to your VNA. There comments for every function in syn that describe what actions the functions perform. You will need to find the command for your VNA that produces that same action, and add it to the function. For example:

```
def polar(model):
    """This action should select the polar display format"""
    commands = {
        Model.HP_8753D: 'POLA'
    }
    return commands.get(model)
```

   In this polar function, the GPIB command sent will change the display format on the VNA to polar form. You will need to add the command for your VNA in the commands block (e.g. `Model.Your_Model: 'YOUR COMMAND'`)
   Repeat this for all of the remaining functions. After You should be set and you will be

able to use your new VNA with the software.

For full, step-by-step details on how each function runs, please go to `vna_comms.py` and `vna_syntaxes.py`. There you will find detailed comments for the code.

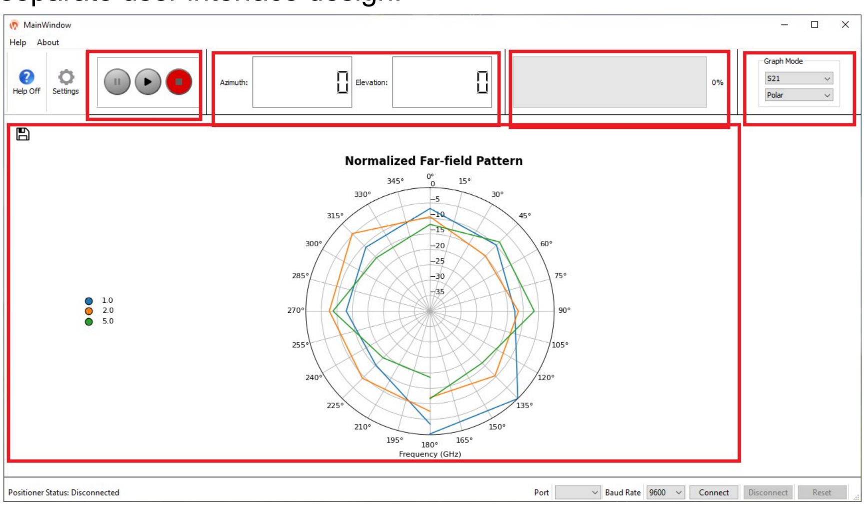
Graphical User Interface (GUI)

With the exception of the top-level code `mainWindow_app.py`, all code files for the GUI are located in the `gui` folder. In this section, we will go over what each file does, and how to modify these files if you wish to do so.

In the gui folder, you will find files with the following names:

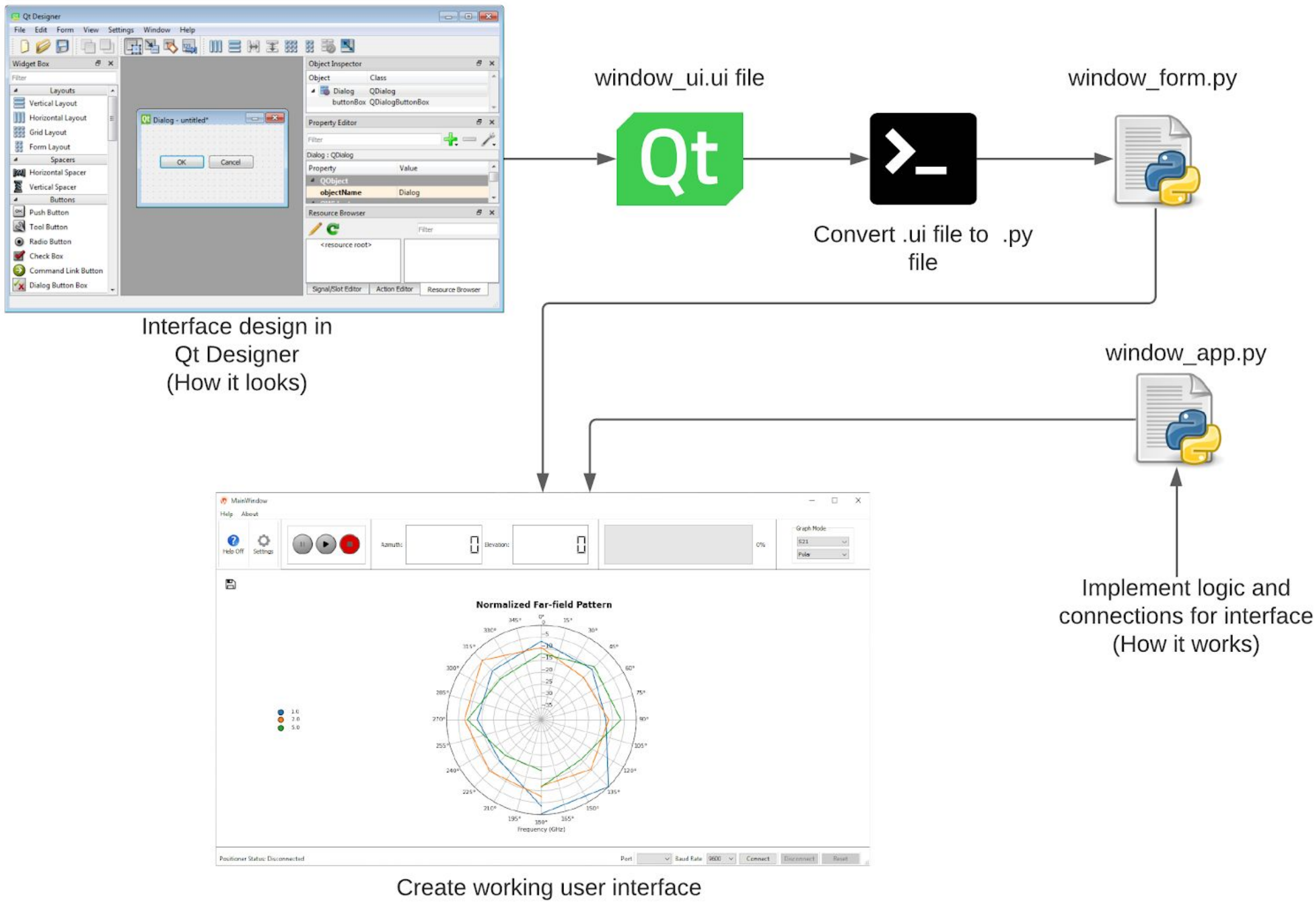
- `[something]_ui.ui`
- `[something]_form.py`
- `[something]_app.py`
- `.png` files

The entire user interface for the software is actually a combination of many smaller subinterfaces. There is an individual interface for the progress bar, the settings window, the graph mode selection toolbar, and so on. In the screenshot below, each red region represents a separate user interface design:



As you can see, this is the reason there are so many files for the GUI.

So what is the process when it comes to working with these files? The figure below shows the general workflow of the user interface design:



First, we use [Qt Designer](#) to design how we want our interface to look. This includes what buttons, text boxes etc. we want to have, as well as where and how big they should be. Once the design is complete in Qt Designer, an `.ui` file is created.

However, due to packaging requirements, we cannot directly implement the .ui file into our program and expect it to work, so this is where the next step comes in. The `[something]_ui.ui` files must be converted into `[something]_form.py` files. **To do this, run the `ui2py.bat` batch file in the** gui **folder, and all `.ui` files will automatically be converted to `.py` files** (you should see the `.py` files being updated, while the `.ui` files remain unchanged). The `.ui` files and the `.py` files contain identical designs for the user interface.

There are also images and icons that are used in the user interface. Again, due to packaging requirements, we cannot directly use the .png files in our program. They have to be converted into binary data and stored in a `.py` file. So how is this done?

- The `resources.qrc` file should include the list of images that we want to convert into binary form.

- Next, run `resources.bat` to compile the images into the binary file. You will need to have PyQt5 installed in order to run the batch file.
- The output is the `resources_rc.py` file. Do not change the name or contents of the file, as PyQt5 will be looking for this specific file when compiling the program.

Now that we've taken care of how the GUI should look, it's time to work on how the GUI will run. This is what all the `[something]_app.py` files are for. They include what happens when certain buttons are pressed (signal triggers), threading the program for the smooth experience, and handling errors within the program.

The file with the most logic is `mainWindow_app.py`. We have tried to make the code as organized and well-commented as possible, but it still may seem confusing if you are not familiar with PyQt and all of its different objects. So here is a quick rundown of some **must-know** knowledge when it comes to PyQt:

- Depending on the type of QObject, they will **signals** and **slots**:
    - A signal emits information, usually when an event occurs (e.g. button presses, the window being resized, when an `emit()` function is called for that specific signal)
    - A signal **connects** to a slot. Think of the slot as receiving the signal. You will see a lot of these `connect` functions in the code. That is where the signals and slots are being established.
    - A slot is usually a function of some sorts, and it is called when the signal connected to it is emitted.
- The settings window and the main window use the QMainWindow class, while all other elements (progress bar, graphing window, etc.) use the QWidget class.
- The `[something]_ui.ui` and `[something]_form.py` files contain the names and properties of the QObject items in the user interface. If you want to implement logic for, say, a specific button, you will need to know the name of that button and what Qt Object type it is. Different QObjects have different properties, signals, and slots.

For more information visit Qt for Python for detailed documentation on every Qt Object that we've used in our source

Data Processing

Found within the data_processing folder is the data_processing.py file. This contains all the code to read the .csv file and plot the data in the appropriate form. The code takes advantage of three Python libraries in addition to PyQt5: Matplotlib, Pandas, and Numpy. Descriptions and use of each library is listed below.

1. **Numpy**
    - Numpy is a library that supports large multi-dimensional arrays and high level mathematical functions.

- The main purpose of numpy in this file is to use their trigonometric functions; i.e. sine, cosine, and radians. These functions were used when having to convert from degrees to radians and when impedance values required the use of cosine and sine functions.
- Numpy arrays were also used a few times to make comparisons much easier across a large data set.

2. **Pandas**
   - Pandas is a library that is useful for data manipulation and analysis.
   - The main purpose of pandas in this file is to read the .csv file into the program, represent the file in an easy to use structure, and isolate specific information to be used in plotting.
   - Reading in data from the .csv file into a pandas.DataFrame,which I will refer to as df going forward, represents the data as a set of rows and columns; similar to the way data can be observed in an excel spreadsheet. With this df, it is possible to access specific frequencies as a set of points and is useful in the plotting portion of the code.
   - Most of the code in data_processing.py is using pandas to create sets of specific frequencies that will be passed into the plotting functions of matplotlib.

3. **Matplotlib**
   - Matplotlib is a library that is useful for its plotting applications and general-purpose GUI toolkits.
   - The main purpose of Matplotlib is to display the data collected by Measurement Control in a form that can be analysed. Matplotlib was also chosen for its integration into PyQt5 applications.
   - Creating a plotting window with Matplotlib starts by creating a MplCanvas object that will contain the figure used to store all subplots. This MplCanvas object is created in the DataProcessing class and is represented by the variable self.sc. Each figure has two subplots that are created in each plotting method (s21_rectangular_plot, s21_polar_plot, s11_rectangular_plot). The first sublot displays the plot with lines (represented as self.sc.ax) and the second contains the legend used to select lines on and off (represented as self.sc.bx).
   - To use the .plot method of our self.sc.ax object two arguments are required but other **kwarg may be added if desired (this can be observed in each plotting method). The first argument is the set of values that represent x or theta. The second argument is the set of values that represent y or rho.
   - The most unique (complex) portion of the data_processing.py file is the MyRadioButtons class that overrides the RadioButtons class. The purpose of this change was to add some functionality that is not built into the CheckBox class provided by matplotlib. Making this change allows the button and its associated line to be the same color. Additionally, when a button is pressed the visibility will turn off or on. When the MyRadioButtons object is created we must pass self.sc.bx as the first argument and an array of strings denoting the lines as the

second. Other **kwargs are passed as well which can be seen in each plotting method.
  - When real time updates are required for plotting, matplotlib has an animate function that acts as a timer. After each time interval, which is currently set to 1000 milliseconds, the program will execute the start_graphing() method.

From a high end perspective, this code is fairly simple. When the MainWindow calls DataProcessing.begin_measurement() the code moves into start_graphing() method and is passed an argument telling DataProcessing whether the plots needs to be updated in real time. If this is a live measurement, the start_graphing() method will continue to execute until measurements are complete.

Below is the basic order of operations of the start_graphing() method:

1. Start by checking if the file was loaded from a run that didn't complete (if so, null values would not exist at eof).If a file was loaded that didn't complete, without running this check the program would think live plotting was necessary.
2. Check to see if the file passed to DataProcessing exists in that location and whether it has data inside.
3. Check if S11 measurements took place inside the file
4. Find the total number of frequencies present in df.
5. Find the largest frequency in the file.
6. If frequencies are in df continue, else return.
7. If the number of frequencies in df are greater than ten, we truncate the df
8. If MainWindow requests an S11 graph
    a. Check if S11 measurements are in df
    b. If S11 measurements are present, we create a new df that only contains those lines, plot, and return (this is a return to MainWindow).
    c. If S11 measurements are not in df, we just return (this is a return to MainWindow).
9. Check if S21 measurements are are in the dataframe
    a. Create a df that only contains S21 measurements
    b. Sort df ascending from frequency, then by phi values
    c. If MainWindow requests polar form, plot and return (this is a return to MainWindow).
    d. Else, MainWindow wants rectangular, plot and return (this is a return to MainWindow).

Note: In a future update, it would be suggested that DataProcessing return a reference to the current figure and/or animation. This would solve the problem of creating and destroying the DataProcessing object each time a new type of plot is requested by the MainWindow. Additionally, the animation function implemented in this version clears the figure and re-plots all data points. A better implementation would be to save a reference to the current data and only add points when new data is placed in the .csv file.

Repackaging the Source Code

Packaging the source code was a complicated process for us, so we have tried to make the repackaging process as easy as possible for you! Running the `build.bat` script by either double-clicking on the file or running it in the command prompt will automatically start the packaging process.

Feel free to read through the `build.bat` script to see what is happening. Here is a recap of all the actions it performs:

- Installs all required python packages with the correct version
- Runs `pyinstaller` and starts the packaging process. It looks into `build.spec` for instructions for the packaging process
- Moves the new executable to the top-level source code directory
- Removes temporary files created during packaging

After a minute a two, the packaging script should be finished. You can copy the `.exe` file and use it anywhere you like (as long as OS and driver requirements are met, of course).

Note: Make sure your modified source code is working properly and error-free in a python environment before packaging it.