# MEEM 4990/5990: Getting Data in the Lab

# Final Project

Due date: see "Components"

## Summary

Design, implement, and test software that drives the experiment or process studied in your interim project. (Note, deviations for your interim hardware design are acceptable, so long as the software and system still meet the final project requirements.) Please note that while the requirements list is long and initially daunting, it can be summed up as "write a clean, easy to use LabVIEW program to acquire data from your system, display it, and save it to a file." Many of the requirements would be difficult to *not* do.

## Components

The final project will be submitted in four parts, each with a separate due date:

1) **Requirements Specification (March 22):** Complete the requirement specification for your project. This is also your grading rubric. Assign points values to each requirement.
2) **Flow/State Chart (March 22):** Create a sketched flow chart of the execution of your program
3) **User Documentation (April 22):** Submit the user documentation (see E.2) by e-mail to jdsommer@mtu.edu
4) **Project demonstration (by appointment during finals week):** Make an appointment to demonstrate your software and system to me during finals week. Demonstrations are expected to last no longer than 30 minutes. During the demonstration I will assign grades to the rubric (Requirements Specification document) you provided.

## A. Requirements Specification

Using the provided template, develop and document a list of requirements for your project.

- The main goal is to think through how the user should interact with the software and how the software should interact with the hardware to perform the experiment or process.
- Given the diverse nature of the projects, it is impossible for me to write a list of requirements that are applicable to every individual project. However, the minimum requirements in this list (the one you are reading right now) must be incorporated into your requirements. To ensure that all of the requirements below are in your requirements specification, a column title "Fulfills Class Requirement(s)" is provided. The class requirements are identified by the lettered and numbered list which you are currently reading (e.g. C.1.4).
- **Ensure that every numbered requirement this list is included in your final Requirements Specification.**
- ***Template items in italics should not be modified or removed.***

- The specification sheet will also serve as your grading rubric. Percent of the grade in major categories has already been assigned (and is in *italics*). You must assign a fraction of the total category points to each specification item. You are free to choose any number of points, so long as the sum of all of your specifications in each category totals the correct number of points for the category.
- Where appropriate, relate the software requirements to your system and hardware requirements by cross-referencing requirements.
- If any of the following requirements are unreasonable or inapplicable to your project, please contact me and we will develop an alternative plan.

**B. Flow/State Chart**

B.1. Develop a flow chart or state chart to that describes your program execution

B.2. To reduce the time-burden of this step, hand-drawn figures are acceptable, but please be sure that your text is legible.

**C. System Requirements (copied from Interim Project)**

C.1. The system to be tested (measured, studied, etc.) must utilize at least three channels total of input and output.

C.2. At least two of the channels must be of different major types (analog input, analog output, digital I/O, counter-timers, instrument control, etc.).

C.3. At least one channel must utilize hardware-timed acquisition

**D. Software Requirements**

**D.1. DAQ: The software shall acquire data from (or generate signals on) each of the channels.**

D.1.1. Show appropriate use of the Mantra: Create, Configure, Start, Read/Write, Stop, Clear

D.1.2. Handle hardware-timed data (either continuous or single-shot "Multiple samples") appropriately using either the waveform data type or an array.

D.1.3. The acquired/generated data must be appropriately synchronized i.e. the program must "know" the time each sample was acquired/generated. Time may be relative (e.g. to the start of the acquisition) or absolute (e.g. May 14, at 2:05:12.012 PM) as appropriate.

**D.2. GUI: The software shall present a graphical user interface**

D.2.1. The interface shall allow for configuration of the experiment, e.g. selection of channels; sensor types; processing algorithms; data acquisition speed, duration, or number of samples; or anything else that makes sense for your project. Consider carefully each parameter and whether or not it should be user-configurable or hard-coded (see also Code Documentation requirements below).

D.2.2. Additionally, the user-interface must provide for information about the experiment/process to be stored with the data files (see Output, below).

D.2.3. The user interface shall be responsive to changes in configuration and execution state. Most likely, the software shall have an event loop.

D.2.4. The interface shall display the acquired data to the user in a prompt and meaningful way. “Prompt and meaningful” will allow the user to determine if the experiment is running as expected as soon as possible.

D.2.5. The interface shall allow the user to select data file output

D.2.6. The interface shall be clean and easy to understand and use (well-aligned, non-overlapping controls and indicators; control and indicators grouped by functional similarity; labels or other front-panel documentation as appropriate, etc.)

**D.3. Analysis: The software shall perform analysis on the input data**

D.3.1. Minimal analysis will include converting the data from voltage to physical units. The use of DAQmx scales, either directly or via NI-MAX channels is permissible, but must be documented in the code.

D.3.2. More complex analysis such as performing math on multiple channels, digital filtering, averaging, etc., is encouraged, as necessary or helpful to the user, but is not required.

**D.4. Output: The software shall save the acquired data in an appropriate format.**
(Alternatively, if generation rather than acquisition is the primary goal of the software, the software shall read data.)

D.4.1. The data format must either be a self-documenting format such as HDF5 or TDMS **OR**, the data must be in a delimited text file which stores attributes and column names in an appropriate header.

D.4.2. At a minimum attributes for the date and time of the acquisition and a brief description of the data must be stored in the file. The user must be able to populate the description field from the user interface.

**D.5. LabVIEW Coding Practices**

D.5.1. SubVIs shall be used collect repeated or similar functionality and modularize code, as necessary or prudent. (Note that it is *possible,* though *unlikely,* that your project will not require subVIs beyond those provided by NI. I am looking for readable, modular code, not subVIs for the sake of subVIs.)

D.5.2. Diagram shall be clean and easy to understand: few wire bends, no wires or nodes hidden underneath other objects, sequential function groups (like The Mantra functions and file functions) kept in a single row when possible.

D.5.3. Appropriate error handling shall be used. All subVI error outputs shall be handled unless sufficient justification is provided in code documentation.

## E. Documentation

### E.1. Code Documentation

E.1.1. Use labels to explain any non-trivial hard-coded values (constants). (E.g., justify a hard-coded sample rate, but you don’t need to justify an array index constant unless you deem it difficult to understand why that constant was chosen).

E.1.2. Use block diagram labels on any portions of the diagram you believe are difficult to understand without such comments.

E.1.3. Document all subVIs using the VI Description (VI Properties->Documentation)

E.1.4. Document all top-level VI (those VIs with which the user interacts) controls and indicators via Tip Strips and/or Description (Right-click->Description and Tip). For any non-trivial or non-obvious subVI controls or terminals, documents these as well.

E.1.5. All VIs should have useful icons

**E.2. User Documentation**

E.2.1. Write a brief user manual for your software.

E.2.2. The manual should discuss configuration settings.

E.2.3. The manual shall walk a user through the typical software use case(s).

E.2.4. Use screen-shots (search for “Snipping Tool” in Windows) to aid the user.

**Demonstration**

I will provide timeslots for appointments during finals week. Each student will make a 30 minute appointment to demonstrate their software and system. During the demonstration, you should first walk me through the system requirements and how your system meets them. Then, you will demonstrate the code. Finally, walk me through the design of the code.

**Grading**

The Requirements Specification document also serves as the grading rubric. As discussed above, you must assign each specification item a point value, such that the values in each section total the value of the section. You are free to assign specifications more or less value as you see fit. Ideally, the specifications you believe are most important for your project will receive the most weight. Note that you will turn in this specification document in advance of finishing the project.

Prior to your demonstration, you must perform a self-evaluation. For each specification, evaluate how well you fulfilled the specification, assigning yourself an appropriate number of points as you see fit. During the demonstration, you will justify your grade to me and I will accept it or adjust it as I see fit.

**Do it**

- ☐ Implement your software according to your requirements
- ☐ Assemble your hardware system at least to a point sufficient to test and demonstrate your software

- ☐ Demonstrate that your software meets your requirements by running it on your assembled hardware system or a reasonable facsimile thereof
- ☐ Deliver your hardware, software and documentation to your adviser/project sponsor

**Suggestions**

- Start now
- Start your VI or copy heavily from already built example file. Use the Example Finder (Help-> Find Examples…). If you browse by task, particularly useful examples may be found in Hardware Input and Output -> DAQmx -> Analog Input.
- Use the Template “User Interface Event Handler”

**Warnings**

- As should be obvious, you may not know everything necessary for your project when this assignment is first handed out. Start anyway. I expect that most knowledge gaps will be filled by lecture but some may require independent research.
- Also, anticipate that your system will have problems outside the scope of the assignment (equipment failure, transducers that do not work properly, etc.). Given the limited time during the semester, try not to spend too much time making the *system* work if possible. Instead, find ways around the misbehaviors that enable you to still develop and test the software.