FYS3240/4240

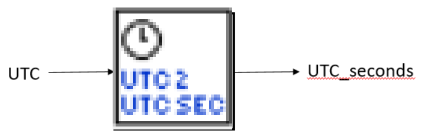
Lab2: **PC-based DAQ with LabVIEW**

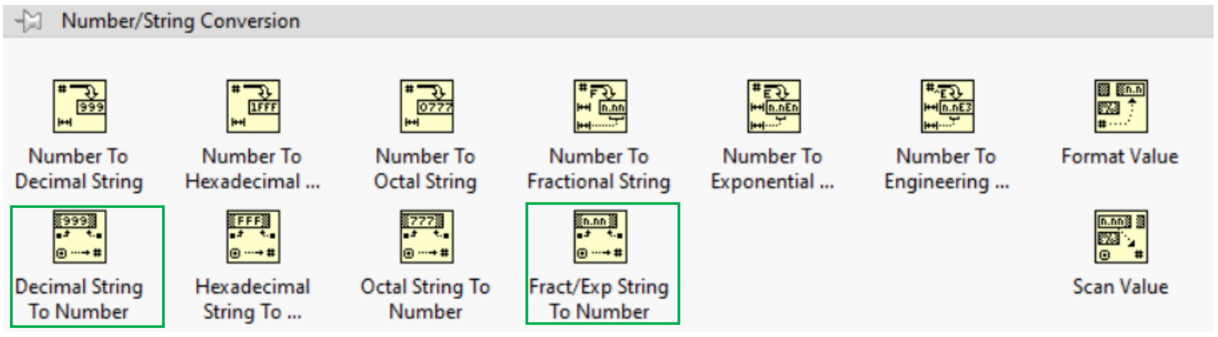
Jan Kenneth Bekkeng | UNIVERSITETET I OSLO, 01.01.2024

.

Exercise 1 – Make a VI to convert from UTC to UTC seconds

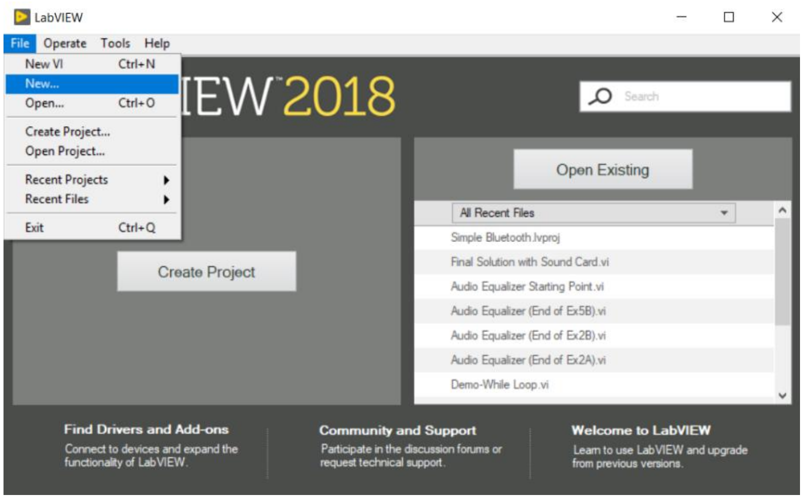
- Make a VI that convert time given in UTC to number of UTC seconds.
  - Save the file as UTC2UTCsec.vi.
- The UTC time should be on the format hh:mm:ss.SSS, for instance 14:52:13.333
- UTC time input can be a Cluster with separate Strings value for hh, mm and ss.SSS. However, to make a more advanced program you could instead use string functions to find the hh, mm and ss.SSS string values from the complete hh:mm:ss.SSS string. *Match Pattern* can be used to find strings before and after the : separation.
- Use *Decimal String to Number* to convert hh (hours) and mm (minutes) to integers (I32) values.
- Use *Fract/Exp String to Number* to convert fractional seconds (ss.SSS) to a double (DBL) value.
- Convert the Decimal values to Double before Add (+).
- Hint: UTCseconds = hh*60*60 + mm*60+ss.SSS
- Make a nice VI icon

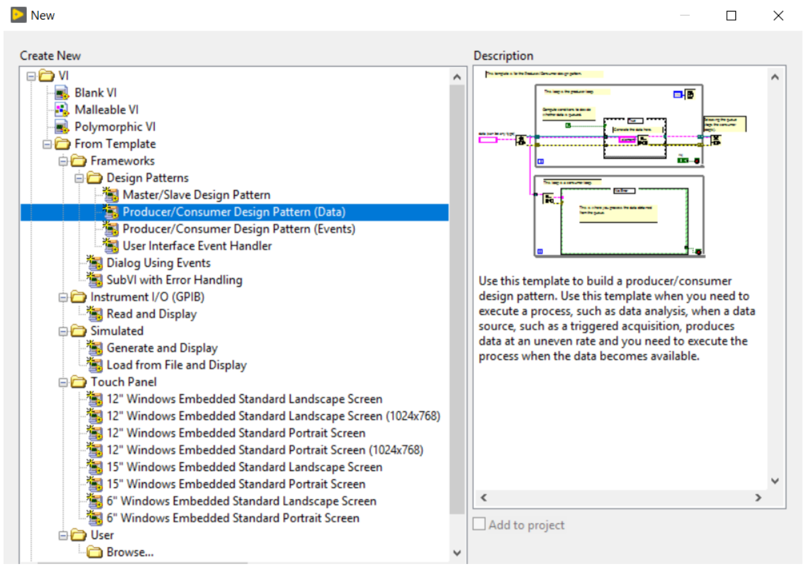



Number/String Conversion
Number To Decimal String
Number To Hexadecimal ...
Number To Octal String
Number To Fractional String
Number To Exponential ...
Number To Engineering ...
Format Value
Decimal String To Number
Hexadecimal String To ...
Octal String To Number
Fract/Exp String To Number
Scan Value

Exercise 2 – DAQ program using producer – consumer template

- Open NI Max and make a simulated M-series NI PCI-6220 DAQ card.
  - Hint: See lab 1
- Close NI Max.
- Open LabVIEW
- Select File – New
- Select Producer/Consumer Design Pattern (Data)
- Click OK
- Save the file as Lab2_ex2.vi.


LabVIEW
File
Operate
Tools
Help
New VI Ctrl+N
New...
Open... Ctrl+O
Create Project...
Open Project...
Recent Projects
Recent Files
Exit Ctrl+Q
IEW 2018
Search
Open Existing
All Recent Files
Simple Bluetooth.lvproj
Final Solution with Sound Card.vi
Audio Equalizer Starting Point.vi
Audio Equalizer (End of Ex5B).vi
Audio Equalizer (End of Ex2B).vi
Audio Equalizer (End of Ex2A).vi
Demo-While Loop.vi
Create Project
Find Drivers and Add-ons
Connect to devices and expand the functionality of LabVIEW.
Community and Support
Participate in the discussion forums or request technical support.
Welcome to LabVIEW
Learn to use LabVIEW and upgrade from previous versions.


New
Create New
VI
Blank VI
Malleable VI
Polymorphic VI
From Template
Frameworks
Design Patterns
Master/Slave Design Pattern
Producer/Consumer Design Pattern (Data)
Producer/Consumer Design Pattern (Events)
User Interface Event Handler
Dialog Using Events
SubVI with Error Handling
Instrument I/O (GPIB)
Read and Display
Simulated
Generate and Display
Load from File and Display
Touch Panel
12" Windows Embedded Standard Landscape Screen
12" Windows Embedded Standard Landscape Screen (1024x768)
12" Windows Embedded Standard Portrait Screen
12" Windows Embedded Standard Portrait Screen (1024x768)
15" Windows Embedded Standard Landscape Screen
15" Windows Embedded Standard Portrait Screen
6" Windows Embedded Standard Landscape Screen
6" Windows Embedded Standard Portrait Screen
User
Browse...
Description
Add to project

Use this template to build a producer/consumer design pattern. Use this template when you need to execute a process, such as data analysis, when a data source, such as a triggered acquisition, produces data at an uneven rate and you need to execute the process when the data becomes available.

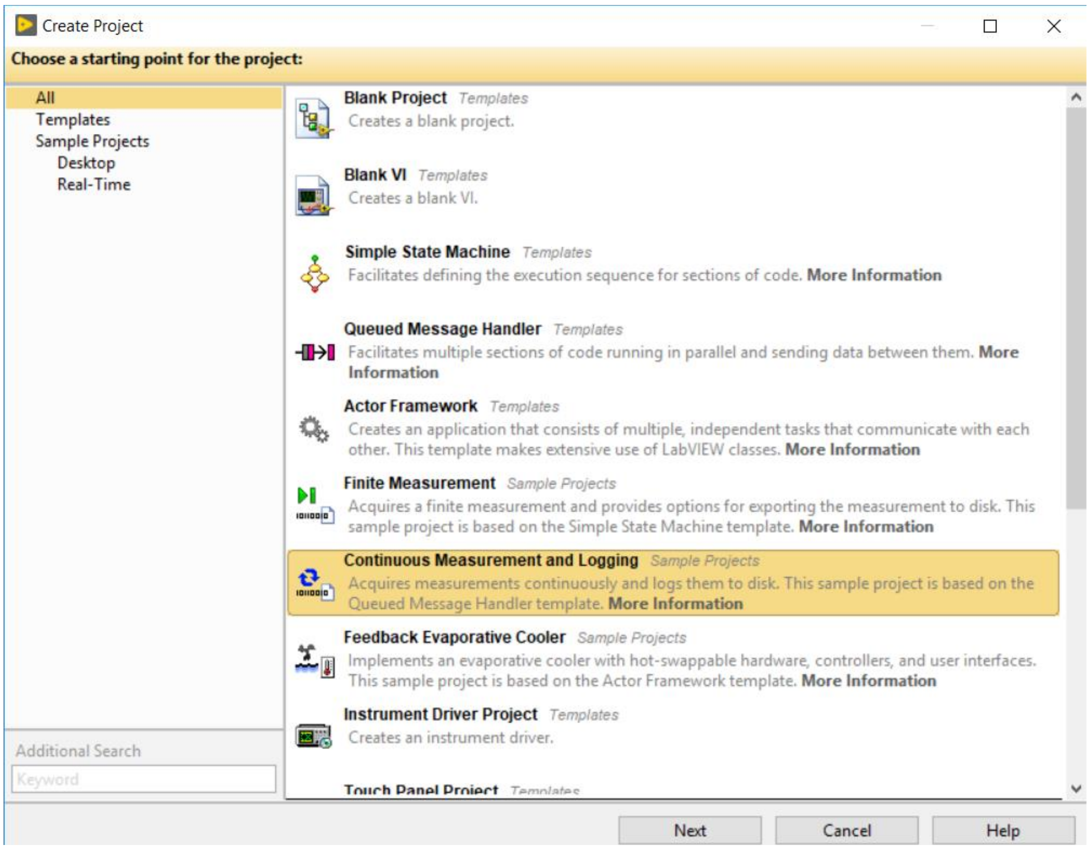
- Make a DAQ setup sub-VI outside the producer loop with the following setup:
  - Two channels (AI0 and AI1)
  - Sample rate of 10 kHz (with on board/internal clock)
  - Continuous sampling with a buffer size of 1000.
  - **Hint:** you could use the **DAQ-assistant and then generate code!**
    - **Configure the DAQ-assistant first, then "Generate DAQmx code"!**
- Read data from the DAQ-card inside the producer loop.
  - Remember that you need a *DAQmx Star Task.vi* outside the producer loop.
- Add data to the data queue in the producer loop, for transmission to the consumer loop.
  - Select 2D DBL as output from the DAQmx read.
  - Remember: the *Obtain Queue* VI must give a unique name to the queue and specify the data type used by the queue (here using a 2D DBL Array Constant).
    - See the LabVIEW lecture for an example of how to make a queue with a 2-dimensional (2D) array of floating point (DBL) numbers.

- Data taken out of the queue in the Consumer loop should be displayed in a **graph** and saved to a **TDMS file** or an ordinary **binary file.**

    - Hint: It is ok to open a file when the program starts (outside the consumer loop) and to close the file (and the DAQ) when you press the STOP button.
    - If saved as TDMS you can look at the file using the *TDMS File Viewer.vi*
- Test your program – what kind of signal do you record?
    - Read back the saved data and make a screenshot.

Exercise 3: DAQ-program using a sample project

- Select the *Continuous Measurement and logging* sample project.
- Click next.
- Give the Project name Lab2_3, select a suitable folder and click *Finish.*
- Open Main.vi.
- Run the program
    - Click *Settings*, and select a file name.
    - Click Star to start the data acquisition.
    - Confirm that you generated data in a TDMS file in the selected location when you press Stop.



Create Project

Choose a starting point for the project:

All
Templates
Sample Projects
Desktop
Real-Time

Blank Project *Templates*
Creates a blank project.

Blank VI *Templates*
Creates a blank VI.

Simple State Machine *Templates*
Facilitates defining the execution sequence for sections of code. More Information

Queued Message Handler *Templates*
Facilitates multiple sections of code running in parallel and sending data between them. More Information

Actor Framework *Templates*
Creates an application that consists of multiple, independent tasks that communicate with each other. This template makes extensive use of LabVIEW classes. More Information

Finite Measurement *Sample Projects*
Acquires a finite measurement and provides options for exporting the measurement to disk. This sample project is based on the Simple State Machine template. More Information

Continuous Measurement and Logging *Sample Projects*
Acquires measurements continuously and logs them to disk. This sample project is based on the Queued Message Handler template. More Information

Feedback Evaporative Cooler *Sample Projects*
Implements an evaporative cooler with hot-swappable hardware, controllers, and user interfaces. This sample project is based on the Actor Framework template. More Information

Instrument Driver Project *Templates*
Creates an instrument driver.

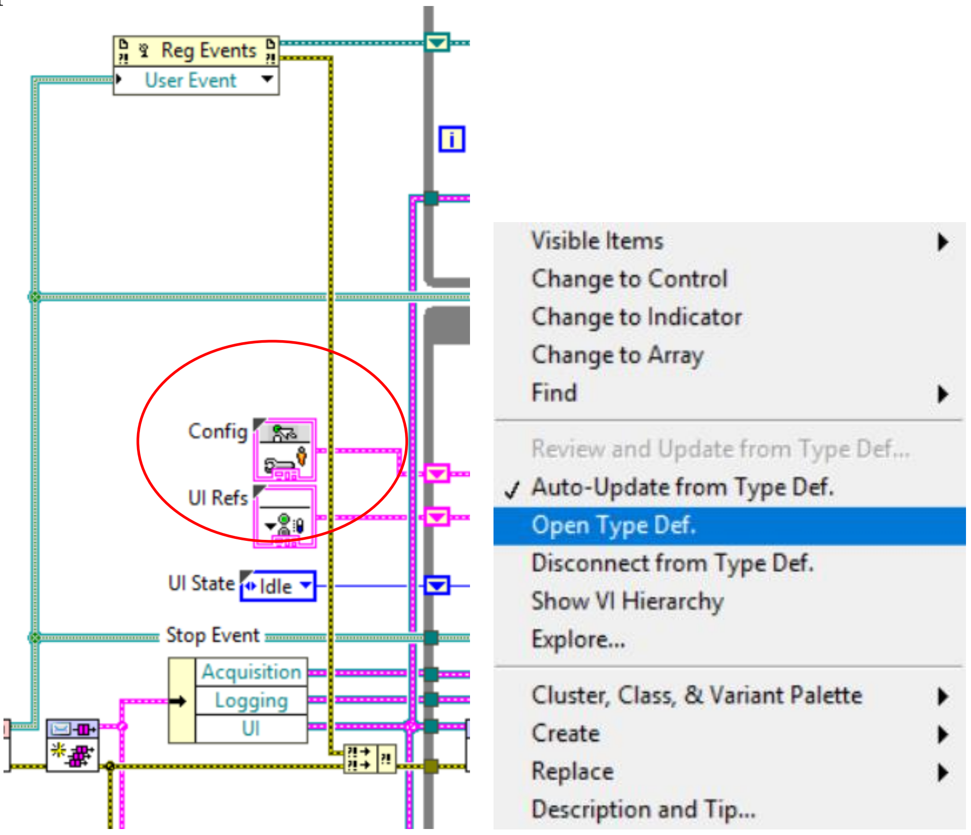
Touch Panel Project *Templates*

Additional Search
Keyword

Next Cancel Help

- Open the block diagram and study the architecture by looking at the top diagram and into the sub-VIs. This is an advanced LabVIEW architecture based on a queued message handler! If you ever are to program a large LabVIEW DAQ-program (for a Master project etc.) this is a good start! Notice that:
    - An event-handling loop is used on the GUI. Based on user selections in the GUI a message is sent to a UI (user interface) message (handler)-loop.

- Separate sub-VIs for acquisition of data (including setup) and storage of data. Both use a state machine inside the sub-VI, and messages (states) and data are transferred using queues.
- Display (graph) of the data is in a separate while-loop. For a more advanced display this should also be in a sub-VI.
- Type definitions (Type Def.) are used, see for instant hardware/file settings (Config) and references to GUI objects (UI Refs). To see this right-click for instant Config and select *Open Type Def*. Type Def are used since they are updated everywhere in the program (also in every sub-VI!) when changed one place.



- Did you find this code difficult to understand? Do not worry, this is advanced LabVIEW programing! And you will <u>not</u> get questions related to the code in exercise 3 on the exam!
- **Challenge:**
  - If you look inside the acquisition loop you will see that a simulated signal is used. Are you able to add the same DAQ-setup (and use the simulated DAQ-card) as you did in Exercise 1? If so, you have made a nice scalable DAQ-program that uses many advanced features!

Hand-in

- **All the LabVIEW code from Exercise 1 and 2.**
- **A screen shot showing the recorded data in Exercise 2**, preferably by reading back the saved data….
- Any code you (optionally) made for Exercise 3.