# A2.4.5 - TWM structure

This report also covers at least partially following activities:

A2.1.1 – Flow chart of TWM tool

A2.1.2 – Extension for a multiple digitizers

A2.1.4 – Concept of the LV to Octave/Matlab interface

A2.2.2 – Integration of the drivers to the virtual driver

A2.4.2 – TWM tool structure

A2.4.3 – Acquisition and control module description

A2.4.4 – Processing module description

A3.3.3 – Guidance on integration of new HW

Following text describes internal structure of the TWM (LabVIEW version).

## References

1. TWM tool, url: <https://github.com/smaslan/TWM>
2. INFO-STRINGS, url: <https://github.com/KaeroDot/info-strings>
3. QWTB toolbox, url: <https://qwtb.github.io/qwtb/>
4. GOLPI interface, url: <https://github.com/KaeroDot/GOLPI>
5. A232 Algorithms exchange format, url:

[https://github.com/smaslan/TWM/tree/master/doc/A232 Algorithm Exchange Format.docx](https://github.com/smaslan/TWM/tree/master/doc/A232%20Algorithm%20Exchange%20Format.docx)

1. A231 Correction Files Reference Manual, url:

[https://github.com/smaslan/TWM/tree/master/doc/A231 Correction Files Reference Manual.docx](https://github.com/smaslan/TWM/tree/master/doc/A231%20Correction%20Files%20Reference%20Manual.docx)

1. A231 Data Exchange Format, url:

[https://github.com/smaslan/TWM/tree/master/doc/A231 Data exchange format and file formats.docx](https://github.com/smaslan/TWM/tree/master/doc/A231%20Data%20exchange%20format%20and%20file%20formats.docx)

## Abbreviations

LV – LabVIEW

CVI – LabWindows CVI

EOS – End of string

DWORD – unsigned 32bit variable

INT16 – signed 16bit integer

INT32 – signed 32bit integer

INT64 – signed 32bit integer

Double – 64bit real number

Cluster – LabVIEW structure of elements

Bool – Logic variable

HDD – Hard drive

TWM – The LV program developed in scope of TracePQM project

GUI – Graphical User Interface

HW – HardWare

QWTB – Q-Wave toolbox [3]

INFO – Brain-dead structured, human readable text file

Matlab – Matlab SW (Mathworks)

GNU Octave – Open source equivalent of Matlab that happens to be almost 100% comatible

m-script – Matlab/Octave’s function file

## Overview

The TWM is organized according to the diagram shown in Figure 0‑1. The whole TWM application consists of two parts:

1. LabVIEW modules (Control and Processing) that controls the instruments, initiates processing and serves as a user interface
2. Calculation or Processing module based on the Matlab/GNU Octave which performs the processing of the acquired data, post-processing and formatting the data for displaying and generation of the measurement report (summary of the results formatted in compact form).

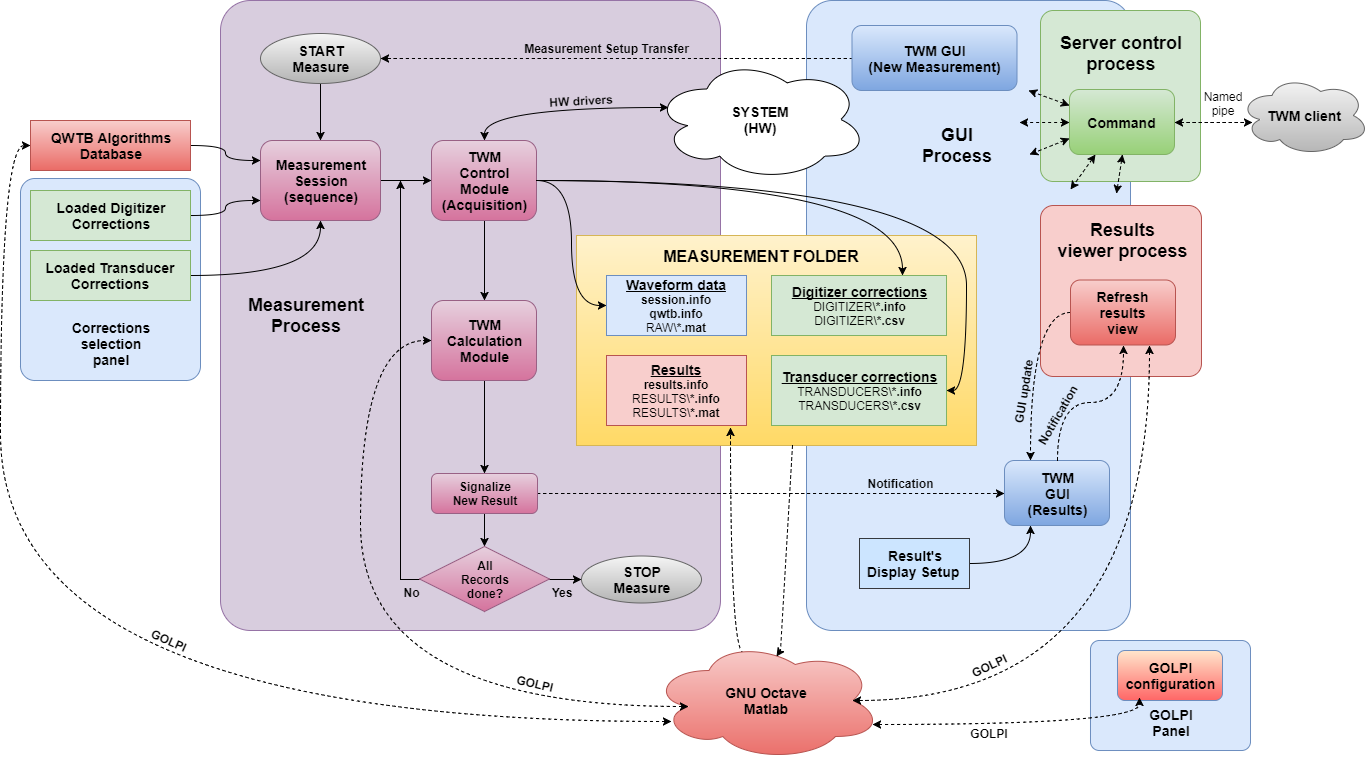


Figure 0‑1: TWM tool structure. The coloured frames are used to distinguish the process in which the tasks run.

The modules communicate on runtime via the GOLPI interface [4] and via files in the measurement folder [7]. So TWM appears as one interactive application. This apparent complication has several benefits. The separation of the acquisition and processing enables several features:

1. The acquired data may be processed at any time. It is possible to just record batch of measurements without processing which may be helpful for time consuming calculations. The processing of the whole batch of measurements can be initiated later either via TWM or on a supercomputer.
2. The same acquired data can be used for calculation of multiple parameters using multiple algorithms.
3. The measurement data is (can be) archived so the data may be reprocessed later if new parameters or correction are needed.
4. The Processing module can run independently on the Control module so TWM can run even without installed Matlab/GNU Octave and the processing can run on any system without drivers required for the TWM (e.g. supercomputer).
5. The Processing module is identical for LabVIEW and CVI version of the tool and the data are interchangeable.
6. The processing module is FULLY transparent. The m-functions of the module do everything: loading the acquired data, loading correction, processing, saving results, loading and formatting results for display, generating report.

The control module is split into four separate processes that run in parallel. Main process is “**GUI Process**”. It contains configuration panels of the HW, configuration panels of the measurement, configurations of the result display and selector of the correction files for the HW components.

When the user wants to initiate a new measurement the “**GUI process**” will create “**Measurement Process**” which does following:

1. Loads correction files.
2. Loads selected QWTB algorithm’s configuration from QWTB alg. database file.
3. Builds measurement sequence.
4. Initiates acquisition.
5. Stores acquired data and full copy of the Corrections and QWTB alg. setup to the measurement folder.
6. When requested by user, initiates processing of the acquired waveforms.
7. Signalizes “new result available” to the GUI process.
8. Repeats from (iv) until all acquisitions are done or user terminates the process by “STOP” button.

When “**GUI Process**” receives notification of the new result or user requires refresh of the results view, it will initiate refresh of the results view according to the current view setup by initiating another process “**Results Viewer Process**”. This process will search the measurement folder and will update the results view or initiates export of the measurement report. Note this process requires Matlab/GNU Octave, because the actual post-processing and formatting is done in Matlab/GNU Octave. The split into the processes means they can partially run in parallel, so when the digitizers are acquiring new waveforms, the “**Results Viewer Process**” can simultaneously perform the post-processing and displaying. The user can even plot graphs of the so far measured results during the measurements.

Finally, TWM contains “**Server control process**”, which allows to control some of the TWM functions and query status and data. The communication happens via Windows named pipe, so it can be controlled from any environment. The key point of this feature is the TWM can be controlled by another application that e.g. performs sequence of measurements. However, note the interface is in development stage and it is not part of the TracePQM project. Thus, it may not be fully developed before end of the project so it will be documented separately when it is ready to use.

## GOLPI

The communication between LabVIEW and Matlab/GNU Octave is ensured by the GOLPI interface [4]. The interface was designed for bidirectional runtime communication between LV and GNU Octave. The communication happens via the pipes which transfers commands and data between the two environments. User can also inspect the communication in console window. The pipes are based on the DLL library “lv\_process.dll” which is part of the project [4]. The “lv\_process.dll” can be used in any language such as CVI. However, it ensures just a low level text data exchange. Variables transfer between the LV and GNU Octave is done at LV level.

The project TracePQM also calls for a communication with Matlab which is far more popular among the potential users. Therefore, the GOLPI library for LV was modified so it also enables almost identic communication with Matlab via the Matlab Script nodes. The nodes are hidden in the GOLPI so from outside there is no difference between use of GOLPI for Matlab and GNU Octave and there should be no difference apart from the performance, which may differ significantly. The only functional difference may be in some algorithms, where Matlab and GNU Octave implementation differs (see algorithms documentation).

### Multi-process GOLPI access

The TWM is a multi-process application. It was not principally possible to ensure the GOLPI is accessed from only one process at the time, so obvious problem arises – the resource sharing between the processes. Obviously only one process can work with Matlab at the time. This is ensured by additional LV library “GOLPI Multi Process.lvlib”. This lib contains several functions. First, before the lib can be used, user must call the VI “golpi\_mpc\_init\_session.vi”, which will initiate the GOLPI instance session.



Figure 0‑2: Initializing VI of the multiprocess library. This must be called somewhere at the beginning of the TWM application before any other access to the GOLPI is made.

This VI initializes the GOLPI instance session, i.e. local variable “GOLPI”, which is part of the “main.vi”. This is only VI that can access this variable directly! All other VIs accesses the “GOLPI” variable via reference. The initialization itself is part of the VI “GOLPI Initialize.vi” whose location in the “main.vi” is shown in Figure 0‑3.

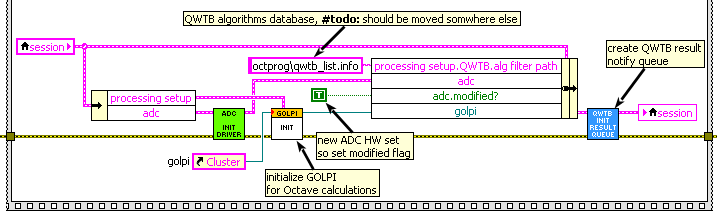


Figure 0‑3: The location of the multiprocess initialization VI.

Before exiting the TWM, the multi-process GOLPI session should be closed by calling VI “golpi\_mpc\_close\_session.vi”. This will also optionally force the GNU Octave process to close, so there is no zombie process left on exit.

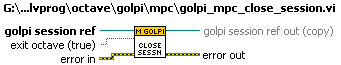


Figure 0‑4: VI for closing the multi-process GOLPI session. The "exit octave" option will force the Octave or Matlab to close (terminate the process).

Whenever any part of TWM needs accessing the GOLPI instance, it must first obtain the GOLPI session by calling VI “golpi\_mpc\_get\_access.vi”. The VI will wait indefinitely for the access to the GOLPI session. That is internally solved by semaphores. It won’t return until all other processes released the GOLPI by “golpi\_mpc\_release.vi” or until the semaphore is destroyed by “golpi\_mpc\_close\_session.vi”. It retunes the local copy of the GOLPI session, which must be store back by the “golpi\_mpc\_release.vi” (see below).

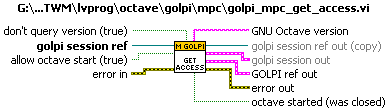


Figure 0‑5: VI for gaining exclusive access to the GOLPI. The "golpi session" is reference to the TWM local variable “GOLPI”. The “allow octave start” option will allow autoamtic startup of GNU Octave/Matlab, when it is not running yet. “don’t querry version” will disable query of the version, which is quite time consuming. The main output is “GOLPI ref out”, which is the GOLPI reference to be used with the GOLPI library VIs. The VI also returns “golpi session out”, which is the multi-process GOLPI session that contains some more elements that may be useful.

When the work with GOLPI is finished, the user must call VI “golpi\_mpc\_release.vi” to store the local copy of “GOLPI ref” (or “golpi session”) back to the TWM variable “GOLPI” and to release the exclusive access to the GOLPI. This will clear the semaphore and thus allow other processes to gain access by “golpi\_mpc\_get\_access.vi”.



Figure 0‑6: VI for releasing the exclusive access to the GOLPI. It accepts

Example of usage of the multi-process GOLPI is shown in Figure 0‑7. The “GOLPI” variable reference is obtained from somewhere (typically from the TWM measurement session “session”). The exclusive access is gained, the GOLPI commands are executed and the exclusive access is released again.

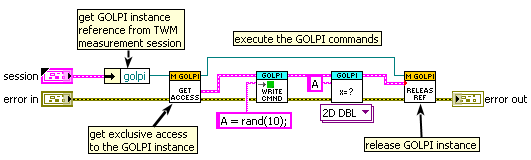


Figure 0‑7: Accessing GOLPI in TWM tool. Before any operation with GOLPI instance, user must obtain the exclusive access. Follow the operations with GOLPI. When done, the GOLPI instance must be released.

## Control and data acquisition module

The Control and acquisition module consists of two sub-modules: (i) Control (user interface GUI), (ii) Acquisition.

This module controls and run the data acquisition by managing initial settings of the sampling process, call to specific instrument drivers through an abstraction layer and handles the acquisition of data from the ADCs, while ensuring storage of the acquired data into the file system.

The controlled of the acquisition process is based on parameters set by the user. Parameters are set prior to data acquisition. Important parameters are filenames, sampling frequency, length of sampling sequence and repetitions. When first called, the module will evaluate the settings, and for certain parameters, the will be prompted with a GUI for confirmation.

During the acquisition, the module updates the values that is visible for the user in the Main window, such as status, the sampling progress, and update of the trace-view and FFT-view if applicable. Finally the data is collected from the instruments and stored to the file system as defined by the parameters.

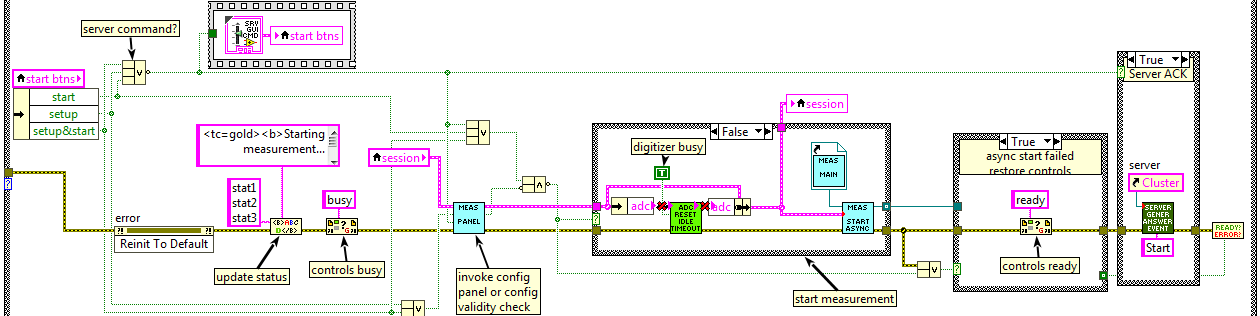


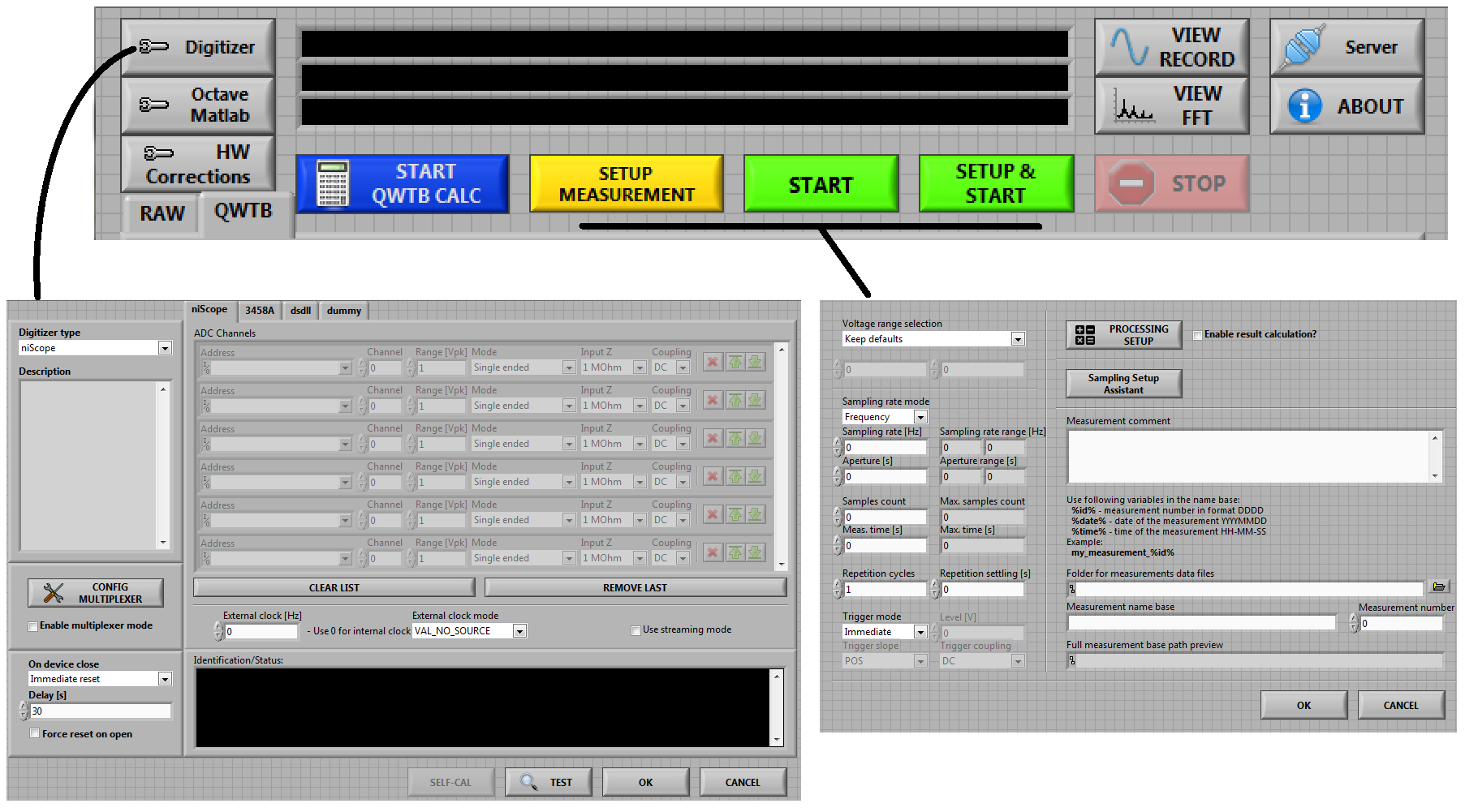
Figure 0‑8: The top-level of this module: Call to the user panel for parameter setting, before Data Acquisition. The acquisition stars by call to the Meas Asyncronious start, which start the main acquisition in a separate process.

### Control module

The Control submodule is a set of subroutines that let user set up the sampling environment and controls the acquisition proses. It also performs validity-check of the parameters before the data acquisition starts, and updates

Part of the control is the user interface. It can be called separately or as part of the start of acquisition. When the acquisition has been selected, a validity-check is done by the control module, and if it finds any issue, the user will be prompted with the GUI for user confirmation.

The acquisition will be performed based on the settings set in the “Meas Config Panel”.



The submodules for the control are:

* “ADC Config Panel”, where specification of digitizer type and related parameters are set, and
* “Meas Panel” for the acquisition the, which are the interface where device none-specific parameters are set. Here things like filename and algorithm selection can be done as well.

Certain parameters are relevant for each specific sample run, and in the main data acquisition these parameters are is set in the “Meas Panel”

**Meas Panel:**

Parameters for sampling and storage is set here, as well as selection of post processing algorithm. The inputs ensure the instrument drivers can be initiated correctly and data is stored at appropriate locations.

The Acquisition is set up by the “Set Measurement” or “Start” option on the front panel.

lvprog\measure\Meas Config Panel.vi

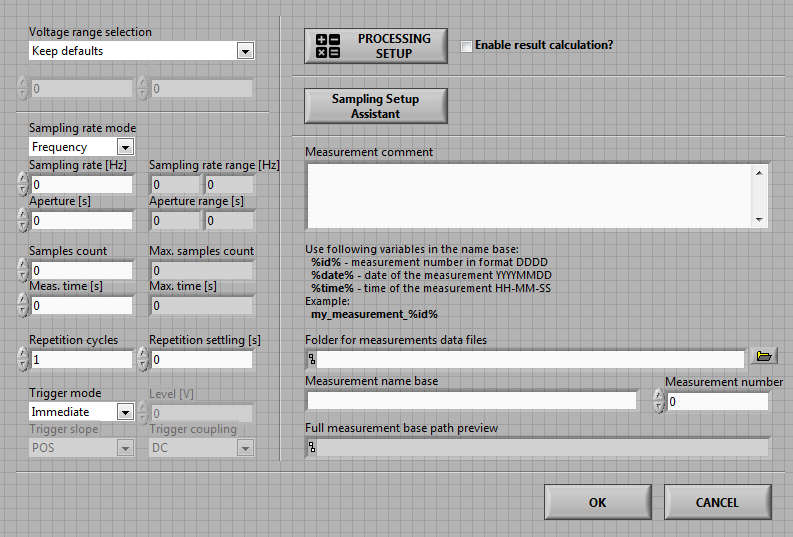
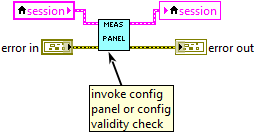
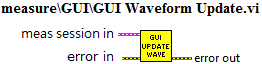
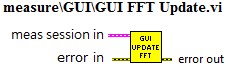


Figure 0‑9: Invoking of the user panel for setting the data acquisition parameters. This is called before the Meas Asynchronous start, which start the main acquisition

Throughout the data acquisition-loop, the front display, data for the View Record and the View FFT is updated with current data. This is done through the “GUI Update Wave” and the “GUI Update FFT”. These two vi’s are called in the acquisition subroutines. (Yellow colour VI-icons)

### Acquisition module

The acquisition module runs in a separate process (see Figure 0‑1).

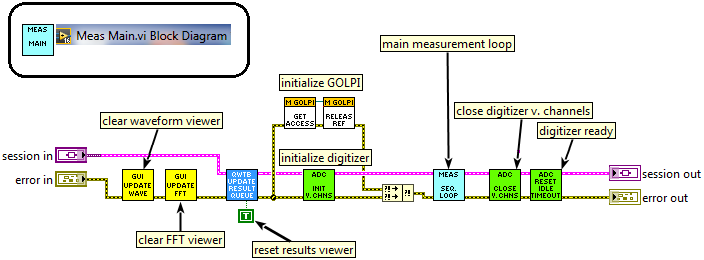
The “main measurement loop” is where the system is set up for data acquisition, and where the system is cleaned up and closed after the sampling is finished. 

Figure 0‑10: MeasMain (details removed for clarity) The overall sequence of the acquisition process. The steps are in short; Initiate, Measure, Close/Reset.

For this module, the following four functions are involved:

* Initialize status fields in the mail GUI
* Initialize digitizer (instrument)
* Call to the main measurement loop (where sampling is done)
* Close digitizer v. channels
* Reset digitizer

**Data acquisition:**

The active data acquisition is done as a subsequence of the “main measurement loop”.

The VI “Meas Loop Sequence” is the outer loop, to accommodate for repetitions.

The steps are in short:

* Set the sampling parameters to the devises
* Open the file system for data storage to File
* Take One record of data from the ADC (data acquisition
* Store data to file system
* Update views on the Main GUI
* (pipe data to the DSP-algorithm)

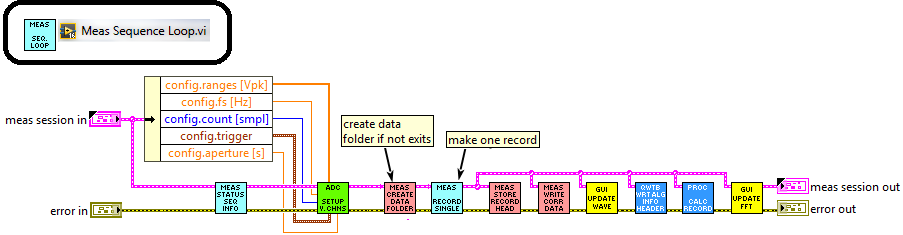


Figure 0‑11: Meas Loop Sequence. Outer loop of the data acquisition sequence

**Inner loop**: Data acquisition. “measure\Meas Single Record.vi”

The core of the data acquisition module is found in this VI. The main task of this VI is to get data from instrument and pipe it into the data stream to the next level, the storage and processing. In addition it do the time critical initiation of the hardware.

Overview of main steps taken by this VI:

* Get the ADC- cofig. (from the control module)
* Opens the MatLab data stream
* Initiate ADC
* Open data stream for the result.
* (Fast-loop for repeated sampling series
  + Fatch data from instrument
  + and write the data to Data stream
* Clean-up ADC after sampling
* Close the Data stream

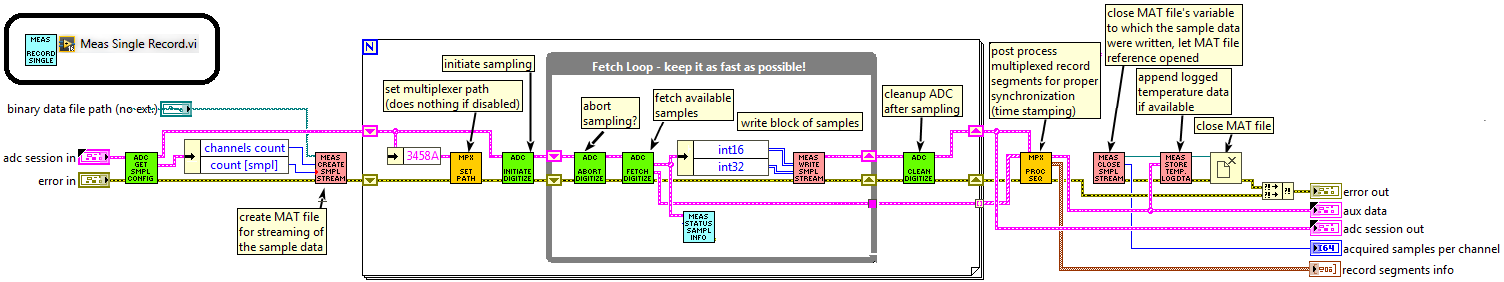
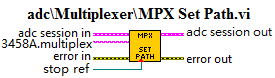
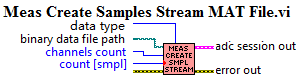
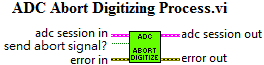
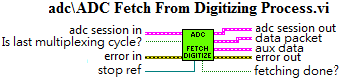
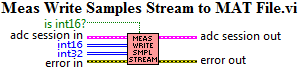
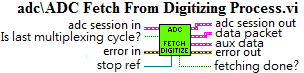


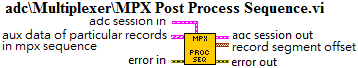
Figure 0‑12: Meas Single Record: The inner loop of the Data Acquisition sequence.



#### Modular driver design

The project objectives call for a modular driver concept. The key idea is the Acquisition module does not access the drivers of the particular instruments directly, because each digitizer requires completely different approach. Therefore the TWM tool would have to use different structure to work with different digitizers. So it was decided to insert a command translation layer in between the acquisition module and the drivers of physical instruments. This layer was called virtual digitizer. All HW specific function calls of each digitizer are translated to a universal format and merged into a few basic VI functions which are, for the acquisition module, identic for any digitizer no matter how different is the HW control implementation inside. The basic block diagram of the TWM in current version is shown in Figure 0‑13.

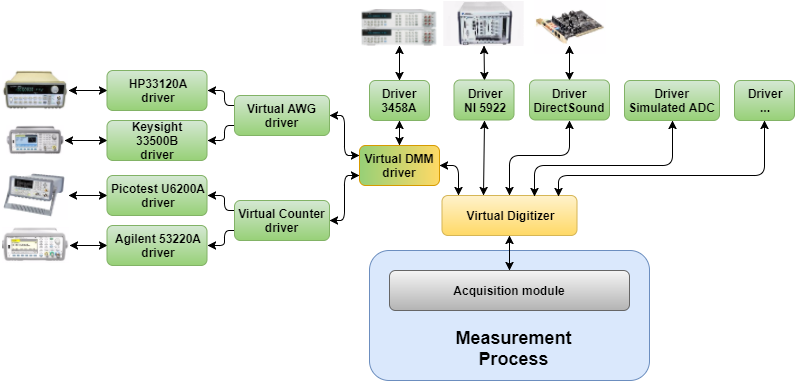


Figure 0‑13: Block diagram of TWM Virtual drivers.

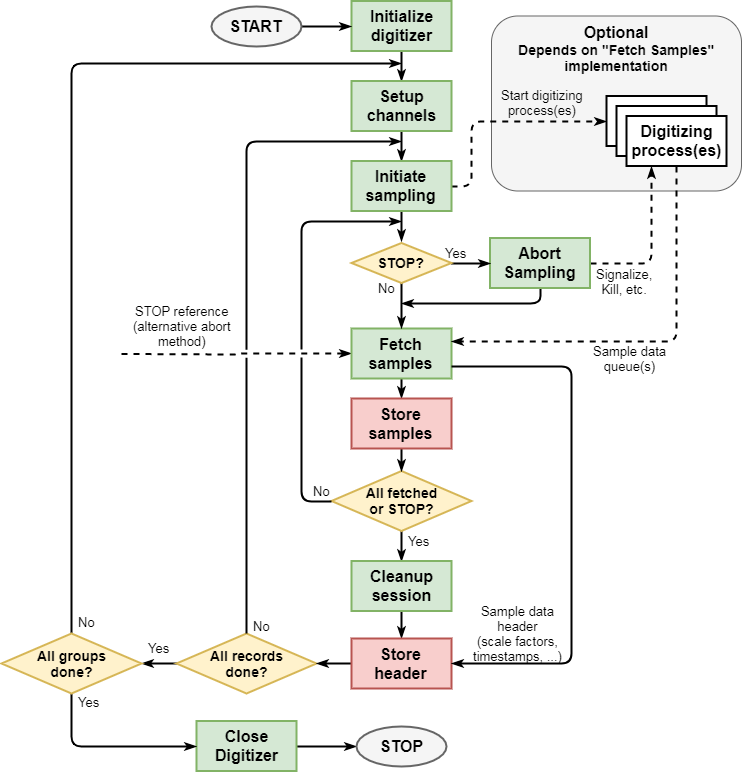


Figure 0‑14: Virtual digitizer driver structure and data flow. Green: virtual driver functions; Red: TWM acquisition module; White: Instrument specific part.

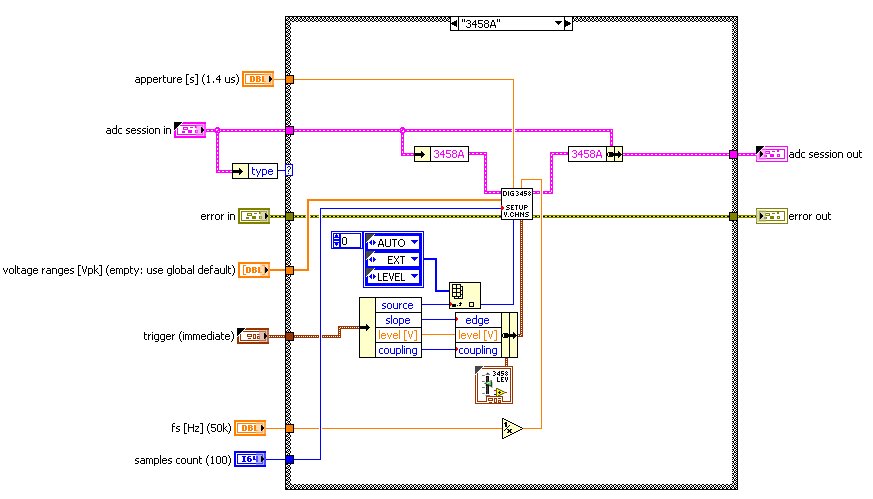
Detailed view on the driver structure and its usage in the TWM acquisition module is shown in Figure 0‑14. The virtual driver functions are shown in green colour. It was decided the driver should not directly write the sample data file, because whenever format of the data changes, each driver would have a different implementation. That is not effective and clean solution. The amount of data in the streaming mode can easily exceeds memory limit, which is just around 1 GB for the 32bit LabVIEW, so the driver cannot simply collect the sample data and then send them to TWM acquisition module at once for saving. Therefore, a rather complex solution capable of runtime storage of sample data based on the background digitizing process(es) was developed. Thanks to the acquisition in the separate process(es) the main fetch loop is non-blocking. Fetching and storing of the sample data runs continuously, so just a limited memory buffers are needed. The acquisition module can easily refresh sampling status and terminate it at any point even if the HW drivers do not allow that directly (e.g. by killing the process(es)). Furthermore, the execution priority of the digitizing process(es) was increased. This way the digitizing runs unaffected by the workload of rest of the application. That may be crucial for the time critical 3458A streaming mode and for high speed streaming from the NI 5922 cards. The throughput was tested and the limiting factor was HDD, which limited the write speed to some 120 Mbytes/s. However, as will be shown in the following chapters, the drivers for other, simpler digitizers do not need to use such a complex structure.

#### Virtual driver functions structure

Following chapters describes particular functions of the virtual digitizer, describes their inputs and outputs, behaviour and also explains where and when are they called by the TWM acquisition module.

To implement a new driver one must adapt and merge the low level instrument driver functions so they principally fit into the functions below (resp. the green coloured blocks in Figure 0‑14). Note there are also a few more functions that need to be implemented apart from the functions shown in the Figure 0‑14.

The virtual driver is a just a wrapper layer that translates the standardized inputs and outputs to the particular instrument drivers. See example for “Setup channels” for DMM 3458A driver:



Each of the following functions contains the case selectors with one item per digitizer “type”. Each function extracts the session related to the particular digitizer (“3458A” in the example) from the virtual digitizer “adc session”, it executes the function(s) of the instrument driver(s) and it stores the modified session “3458A” back to the virtual digitizer “adc session”. Other functions are made the same way. The only exception is the configuration panel for the digitizers which will be described separately.

There are just a few steps to be done to integrate new drivers. First, change the type definition of “type” in the “adc session”, i.e. add a new item to “type” Enum. The item names should be chosen clearly, such as “NI 9234”. Next, a session cluster (or class) of the new driver must added to the “adc session”. This object can contain absolutely anything. It depends on the driver. Finally, each of the case selectors in each of the virtual driver functions must be extended by the new page, e.g. “NI 9234” and the driver functions must be inserted. TWM will then automatically allow to use the new digitizer without any changes in the rest of the application.

##### ADC session

“ADC session” is a virtual digitizer cluster that has to contain all sub-sessions of the particular digitizers. It also contains several common items. Details on content of this cluster at the time of writing this document (may extend in future):

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| niScope | cluster | 5922 (niScope) driver session. |
| 3458A | cluster | Virtual DMM 3458A driver session. |
| Dsdll | cluster | DirectSound driver session. |
| simadc | cluster | Simulated ADC session. |
| \* | \* | New driver sessions… |
| Type | enum | Selected digitizer type {‘niScope’,’3548A’,’DirectSound’,’dummy’}. |
| modified? | bool | Flag set by TWM to “True” when HW setup was modified. |
| channel idn str | 1D array of string | Array of last queried identifier strings of particular channels of selected digitizer. One item per channel. |
| aux instr idn str | 1D array of string | Array of last queried identifiers of auxiliary HW related to the selected digitizer (e.g.: AWG, Counter, etc.). |

Adding a new digitizer means the session of the digitizer driver will be added and “type” enum will be redefined to contain unique identification name of a new digitizer. The rest must not be changed.

#### Virtual driver function reference manual

##### Initialize driver (optional)

Some digitizer drivers may need to perform some step to make them usable. This optional function is called once automatically on the TWM startup.



Function inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |

##### Enumerate devices (optional)

This optional function is called manually in the digitizer selection panel. It was added, because some of the drivers may require additional manual refresh of the installed HW configuration. It was used for the DirectSound drivers where it enumerates available input capture devices.

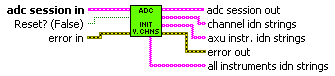


Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| capture devices ring ref | in | reference to ring | Reference to a ring control to be filled with enumerated devices. |

##### Initialize digitizer (required)

It is first function called by TWM before new measurement. Its purpose is to initialize and identify all HW components related to the digitizer. E.g.: for virtual digitizer based on the 3458A multimeters it is one or more 3548A units and optionally a pulse generator AWG or a counter. The function also sets the parameters which are not expected to change during the whole measurement session, such as mode of sampling (“DC V”, “DSDC”, …), coupling, etc. It always returns unique and clear identifiers of the channels and auxiliary HW.

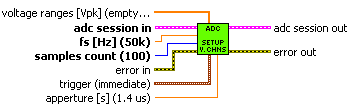


Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| reset? | in | bool | Force reset of the instruments? Similar to standard instrument driver template. |
| channel idn strings | out | 1D array of strings | Queried identifiers for each virtual channel. |
| aux intr. idn strings | out | 1D array of strings | Queried identifiers of additional HW related to the selected digitizer. |
| all instruments idn strings | out | 1D array of strings | All identifiers merged to one array. |

##### Setup channels (required)

This function is called once per group of measurements. I.e. it is not recalled before each repetition cycle so the driver must be prepared to perform several acquisition without recalling this. It will configure the virtual channels of and the virtual digitizer to the desired setup prior the acquisition. This function sets sampling rate, sample count, aperture, triggers, ranges, etc. The digitizer shall be ready to start acquisition after this function call.



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| fs [Hz] | in | double | Desired sampling rate in [Hz]. |
| samples count | in | double | Desired samples count per channel. |
| trigger | in | cluster | Trigger setup cluster. |
| aperture [s] | in | double | Desired aperture of the ADC. Note this parameter will be ignored if digitizer does not support it. |

Cluster “trigger” contains following items:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| source | Enum | Trigger type {‘Immediate’ – start immediately; ‘External’ – from external trigger input; ‘Level’ – input channel level trigger}. Note the ‘Level’ is always related to the first channel. This may be eventually configured in driver specific configuration panel. |
| slope | Enum | Trigger slope sensitivity for “Level” and “External” triggers {“POS” or “NEG”}. |
| coupling | Enum | Coupling of the “Level” trigger {“DC” or “AC”}. Eventual other configurations must be handled by the driver itself and set from configuration panel. |
| level [V] | Double | Trigger level for “Level” mode in Volts. |

##### Initiate sampling (required)

This is when TWM is ready to digitize. This function should immediately initiate the sampling (arm the virtual digitizer) and return. The actual operation depends on the implementation of the “Fetch samples” function (see below).



Input and outputs:

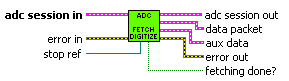
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | In | cluster | Error signal. |
| error out | out | cluster | Error signal. |

##### Fetch samples (required)

The function is called in the loop to fetch the sample data and status. Purpose of this function is to obtain the acquired data from the virtual digitizer channels. There are three basic options for its implementation depending on the particular digitizer:

1. Blocking function that won’t return until all samples were acquired. This is suitable for smaller counts of samples, however the cost for this solution is TWM cannot query state of the sampling and the termination by STOP command is harder to implement (or impossible) as well as timeout. This is typical way the most of the instrument drivers are made.
2. Asynchronous function that just checks weather the sampling is done and eventually returns available samples block. When not done, it will just return status if possible. This way the sampling loop is not blocked and the TWM can update sampling status and easily terminate sampling by STOP command. This mode is however not possible for all digitizers as some of them do not allow asynchronous operation.
3. Complex implementation shown in the Figure 0‑14 where the “Initiate sampling” just starts the “digitizer process(es)” and the “Fetch samples” periodically checks, weather there are a new sample data available. If so, it returns block of samples that is stored to the file by “Store samples”. It is hard to implement, but it seems to be very useful for the 3458A streaming driver and for high speed PXI 5922 driver because the process(es) may be run with increased priority. This should prevent overflows for high speed digitizing or time critical digitizing (3548A).

Note this function always receives LV reference to a global Boolean variable “STOP”. This reference can be used as an alternative way to terminate the sampling (default is “Abort Digitizing Process”). Note the driver function must not change the values of “STOP”. It is just for reading.



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | Cluster | Virtual digitizer session. |
| adc session out | out | Cluster | “adc session in” copy with eventual changes. |
| error in | in | Cluster | Error signal. |
| error out | out | Cluster | Error signal. |
| stop ref | in | Reference to bool | Reference to the global Boolean variable STOP. The variable becomes “True” when stop is requested. The function cannot modify the flag. |
| data packet | Out | Cluster | Cluster with block of sample data. |
| aux data | Out | Cluster | Cluster of additional data returned by the driver. |
| fetching done? | Out | Bool | This flag must be “True” when sampling is finished or it was terminated. |

The “data packet” is a cluster containing following:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| int32 | 2D array of int32 | 2D array of samples. One column per channel. |
| int16 | 2D array of int16 | 2D array of samples. One column per channel. |
| is int16? | Bool | Defines which of the “int16” or “int32” is valid. The other must be empty. |
| all done? | Bool | “True” when all samples were fetched. |
| sampling? | Bool | “True” when digitizing is in progress. |
| valid? | Bool | “True” means the other items are valid. Otherwise they are ignored by acquisition module. This may indicate the iteration of fetching was returned no data. |
| instr buffer [%] | Double | Indicates utilisation of the digitizer buffer. This is e.g. used for the 5922. May be “NaN” is not supported. |
| queue buffer [%] | Double | Utilisation of the data queue between digitizing process(es) and fetch function. May be “NaN” if not supported. |
| offset [smpl] | Int64 | Offset of the first sample in the block from start of the acquisition. Counting from zero. |
| count [smpl] | Int64 | Samples count in the sample array per channel. May be zero if no data fetched. |

The “aux data” content:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| T smpl | 1D array int64 | Indices of the samples to which the temperature readings are aligned. |
| T [deg C] | 2D array of doubles | 2D array of temperature readings during the acquisition. One column per channel, one row per item of “T smpl”. Note this is optional and the arrays may be empty. |
| Time stamp [s] | Double | Relative timestamps returned by the channels. These are relative time intervals in Seconds of the first sample of each channel related to some common event, e.g. reset of 5922. |
| Gain [V] | 1D array of doubles | Gain factors to get voltage from the integers in data packets. One per channel. |
| Offset [V] | 1D array of doubles | DC offset to add to real samples to get actual voltage. One per channel.  u(k) = gain\*y(k) + offset; u – voltage, y – integer sample |
| Increment [s] | Double | Sampling period [s]. |
| Valid? | Bool | “True” means the other items are valid. Otherwise they are ignored by acquisition module. Note the driver may return this cluster valid any time during the sampling. Whenever the flag is set, acquisition module remembers the data. So it does not matter if it returns at the start or the end of sampling. |

##### Cleanup session (required)

This function should terminate everything that may have left in the memory/system after the “Initiate sampling” function, e.g.: the processes, queues, shared memory, etc. This is called by TWM every time to cleanup after acquisition (even terminated or failed).



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |

##### Close digitizer (required)

This function is called after acquisitions are done even in case of error or termination. This function should put all affected instruments to some default safe state and close opened sessions to them. It is strongly recommended to put the instruments to such a state so they cannot be damaged. E.g.: not 50 Ω input, higher range, etc. Also it is good practice to turn all instruments programmatically to the local control as some of them may not even have “Local” button.



Input and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| Reset to preset mode (True) | In | Bool | “True” means the function should reset the digitizer instruments to some safe default state before closing the handles. |

##### Abort Digitizing Process (recommended)

This function is called in the fetch loop when GUI signalizes STOP command. Implementation depends on the “Fetch samples” variant. For variant (i) it cannot be used as the function is blocking. For the other two variants, it may either send the signal to the digitizer if it supports such a function, or it can kill the digitizing process(es) (variant iii). Naturally “Fetch samples” must be able to recognize the digitizing process(es) were terminated and also signalize sampling done so acquisition module will leave the fetching loop.

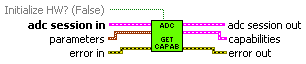


Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| Send abort signal? | In | Bool | Sends abort only when “True”. Nothing happens when “False”. |

##### Get Digitizer Capabilities (required)

This function returns capabilities of the selected digitizer. It is called at various places of the TWM. It should NOT touch the HW by itself! All HW related information shall be obtained in the “Initialize digitizer” function and kept in the digitizer session even after “Close digitizer” is called! TWM decides by itself when to call “Initialize Digitizer”+”Close digitizer” to refresh the parameters so the time consuming querying is not performed when it is not needed. The capabilities are used to limit the GUI entries and disable the unsupported features. The driver should just query the information from the session and return the desired capabilities.



Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | in | cluster | Virtual digitizer session. |
| adc session out | out | cluster | “adc session in” copy with eventual changes. |
| error in | in | cluster | Error signal. |
| error out | out | cluster | Error signal. |
| Initialize HW? | In | Bool | Forces new query of the instruments capabilities. Otherwise uses last queried capabilities from digitizer session (fast mode). |
| Parameters | In | Cluster | Cluster of some parameters that may be needed to obtain the capabilities. |
| Capabilities | Out | Cluster | Cluster of obtained capabilities. |

Cluster “parameters” contains following:

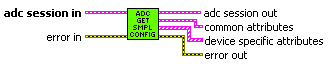
|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| fs [Hz] | Double | Current sampling rate in [Hz |
| aperture [s] | Double | Aperture time [s]. |

Cluster “capabilities” contains following:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| Max samples count | Int64 | Maximum number of samples to be acquired in one record. |
| Max fs [Hz]  Min fs [Hz] | Double | Maximum and minimum sampling rate [Hz]. |
| Ts step [s] | Double | Available sampling period step in which the rates can be set. |
| fs step [Hz] | Double | Available sampling rate step that can be set by digitizer. Note either “Ts” of “fs” step can be used. The other must be “NaN”. |
| Smpl rate step mode | Enum | Mode of sampling rate selection {‘const period’,’const frequency’}. Selects which of the “Ts” of “fs” is valid. |
| Max Ts [s]  Min Ts [s] | Double | Maximum and minimum sampling period [s]. |
| Aper min [s]  Aper max [s] | Double | Minimum and maximum apertures [s]. Or “NaN” if not supported. |
| Channels count | Double | Number of virtual channels available configured. |
| Allows streaming? | Bool | Set when the driver supports two modes: memory buffer and direct streaming. If it supports just one, it is ignored. |
| Streaming on? | Bool | Streaming mode configured. |
| Has level trig? | Bool | Driver/digitizer supports level triggering. |
| Has ext trig? | Bool | Driver/digitizer supports external input triggering. |
| Has aperture? | Bool | Driver/digitizer supports setting the appertures. |
| Has ranges? | Bool | Driver/digitizer can set multiple ranges. |
| Has temperature? | Bool | Driver/digitizer supports temperature measurement. |
| Has temperature log? | Bool | Driver/digitizer supports temperature logging during acquisition. |
| Has selfcal? | Bool | Driver/digitizer supports self-calibration routine. |

##### Get Current Setup (required)

Similar to the “Get Digitizer Capabilities”. It should not touch the HW. It should return last used configuration from the digitizer session. This function returns two groups of parameters. First, the standard ones, e.g.: sampling rate, samples count, trigger, etc. Next, the specific for given digitizer.



Function inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| adc session out | Out | cluster | “adc session in” copy with eventual changes. |
| error in | In | cluster | Error signal. |
| error out | Out | cluster | Error signal. |
| Common attributes | Out | Cluster | Standard attributes/parameters of the digitizer. |
| Device specific attributes | Out | 1D array of clusters | 1D array of clusters containing:  “name” – attribute name string  “value” – 1D array of test string with formatted value  “is constant per group” – when trues, the attribute is stored just once for measurement group. Otherwise it is stored for each record.  Note the value may be numeric. In that case format the number to decimal, floating or exponential format with decimal dot “.”. These attributes are not used by TWM anywhere, they are just store as additional items to the measurement session header. They will appear there as “name:: value” or as:  #startmatrix:: name  value(1); value(2);…  #endmatrix:: name  Note the “aperture” value is one of these attributes. |

Cluster “common attributes” contains following items:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Meaning** |
| Count [smpl] | Int64 | Configured number of samples to acquire per channel. |
| Channels count | Int32 | Number of configured channels in the virtual digitizer. |
| fs [Hz] | Double | Configured sampling rate in [Hz]. |
| Ranges [V] | 1D array of doubles | Array of set range values as defined by the particular drivers. One value per virtual channel of the digitizer. |
| Trigger | Cluster | Trigger setup, see above. |
| Ext freq. locked? | Bool | Status of PLL lock if supported. |
| Streaming on? | Bool | Set when streaming is enabled. |
| Is int16? | Bool | Set when data is/will be in int16 format for the configured sampling setup. |
| Bitres | Int32 | Actual bit resolution (how many bits are utilised in the integer). |

##### GUI Get Info (recommended)

This function takes digitizer session and returns a brief description of the digitizer which is displayed in the digitizer panel. It must not touch the HW. It may contain e.g. trigger connection notes (3458A mode). It is called in the digitizer configuration panel.



Inputs and outputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| Configuration description | Out | String | String with brief description of the current configuration of the digitizer. |

##### Selfcal Virtual Channels (optional)

This function should initiate self-calibration of the digitizer HW components if such function is supported. It is synchronous operation. TWM is blocked during its execution.

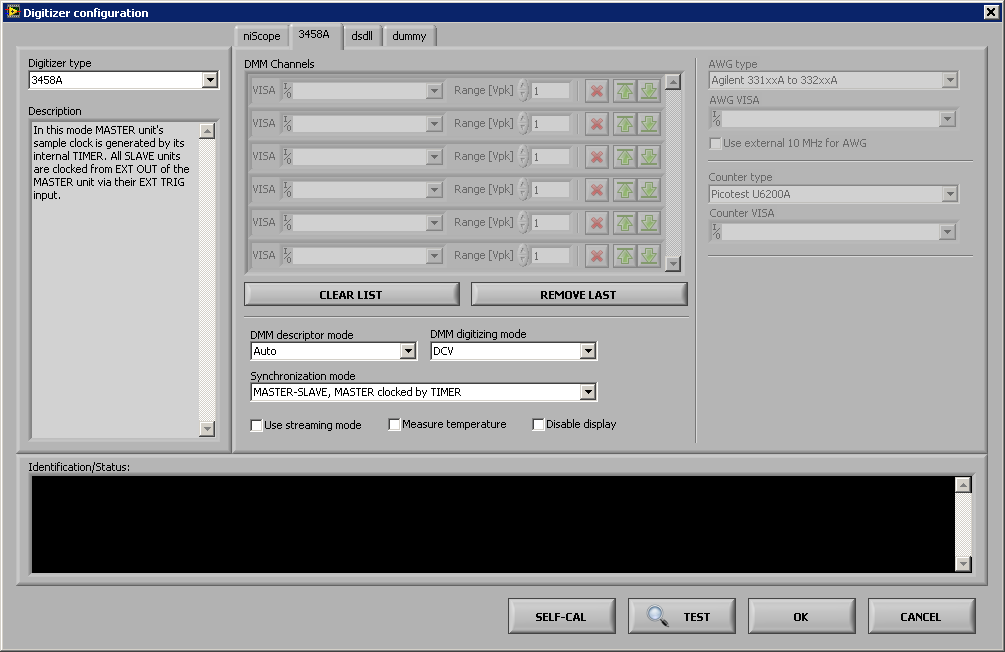


Inputs and outputs:

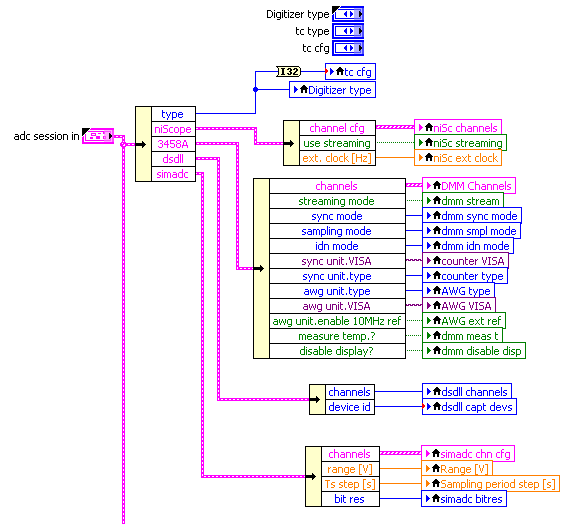
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Meaning** |
| adc session in | In | cluster | Virtual digitizer session. |
| adc session out | Out | cluster | “adc session in” copy with eventual changes. |
| error in | In | cluster | Error signal. |
| error out | Out | cluster | Error signal. |

##### Digitizer configuration panel (required)

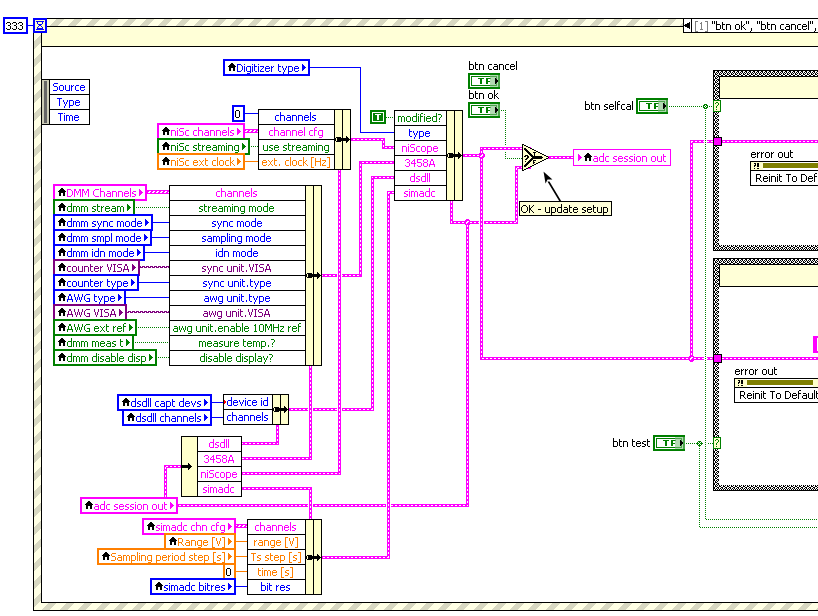
The digitizers in the TWM must be configured before they can be used. The configuration is done in a panel “Digitizer configuration”. The panel contains page control with one page per digitizer:



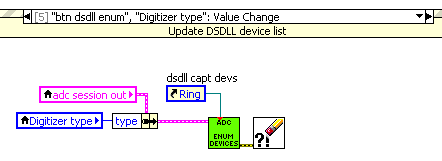
A new page must be added to implement a new digitizer. The order of the pages must match the order in the “type” Enum in the “adc session”. The new page can contain any configuration needed for the new digitizer. On initialization the current settings from the digitizer sessions must be set to the panel items related to the particular digitizers:



Accordingly before exiting the panel, the new settings must be stored back to the “adc session”:



The panel itself is based on the Event structure. Most of the events are common for all digitizers and thus doesn’t need to be modified. However, each digitizer may have some special requirements which must be placed in the new event. Example for DirectSound driver:



## Processing module

Processing module consists of two components: (i) LabVIEW VI’s and (ii) Matlab/Octave functions. The Matlab functions are common for the LabWindows/CVI version of the tool and they can be used standalone without TWM.

### Processing module – LabVIEW component

The LV component of the processing module consists of several parts: (i) Processing configuration GUI; (ii) Algorithm execution routines; (iii) Results viewer/interpreter; (iv) Batch processing GUI. The TWM allows two modes of processing. The first and simplistic is an execution of user entered m-code as it is and simple display of eventual calculated results. This is intended just for debugging purposes and simple tests and it won’t be described to details. The relevant mode is execution of a QWTB algorithm [3]. In this mode, the TWM interacts on runtime with the QWTB m-functions via the GOLPI interface [4] to obtain the information about available algorithms or execute them or to retrieve the calculated results. Logical flow of the processing module in runtime processing during measurements is shown in Figure 0‑15. Note the Matlab does not return any data back to the LV (apart from error messages). The results are stored back to the measurement folder and queried asynchronously by the TWM tool whenever requested.

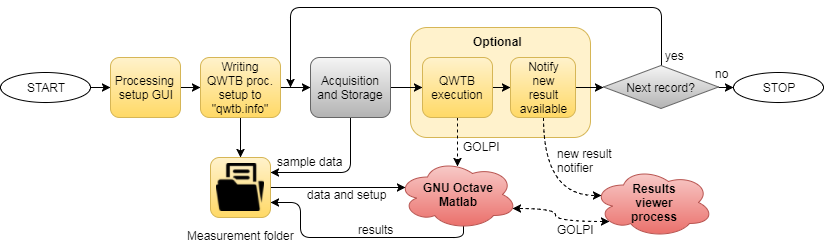


Figure 0‑15: Processing module logic flow. Note the grey blocks are not part of the processing module.

#### Processing configuration GUI

The processing GUI is a panel dedicated to configuration of the new calculation. It is shared for the runtime processing of the sampled data and also for the batch processing of previously sampled data. It has several sub-functions.

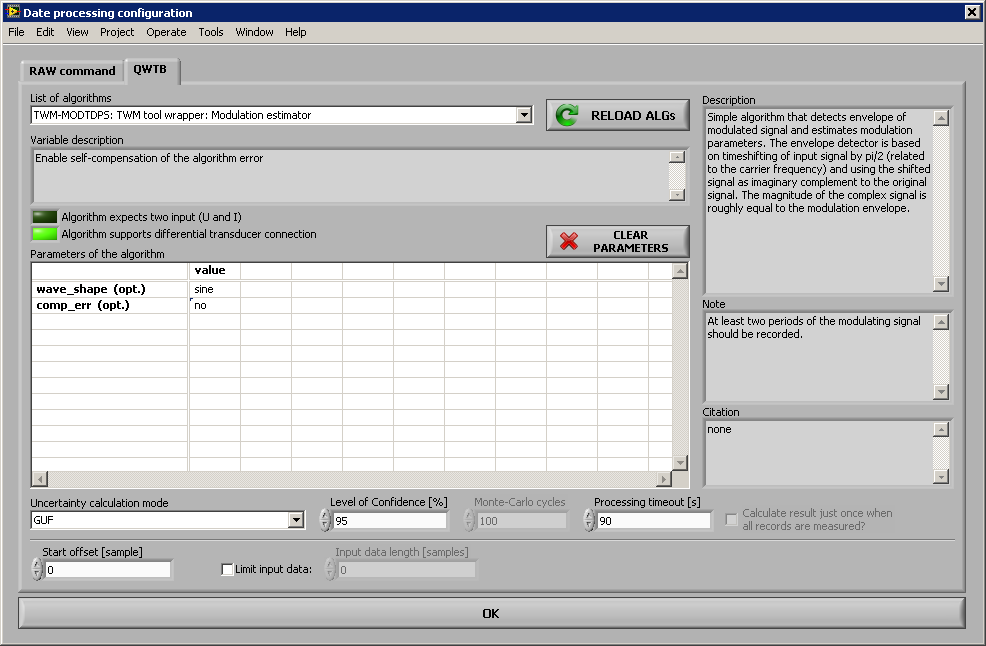


Figure 0‑16: GUI panel of the QWTB processing setup.

First, it queries the list of available algorithms by calling an m-function “qwtb\_load\_algorithms.m” via the VI “Meas Proc QWTB Load List of Algorithm.vi”. The obtained algorithms are displayed in the selector. The function also applies the filtering of the algorithms based on the content of file “qwtb\_list.info”. This is needed to prevent user of TWM from selecting QWTB algorithms that are not compatible with TWM.

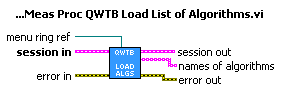


Figure 0‑17: Loader VI of the QWTB algorithms. “session” is the measurement session of TWM and the “menu ring” is a reference to the control that receives the list of available algorithms.

Next, the GUI allows to select the algorithm from the list and query its options and description. This is done by calling the m-function “qwtb\_load\_algorithm.m” via the VI “Meas Proc QWTB Load Algorithm.vi”. The GUI will obtain the standard QWTB info entries, such as the full name, brief description and notes. It will also query the TWM specific flags which tell TWM if the algorithm can accept differential input sensors, if it is algorithm with multiple inputs (e.g. power) and if it can process multiple records at once. It also queries available modes of uncertainty calculation so user can select only the valid ones. The function also queries the list of algorithm’s user parameters and displays them in the parameter matrix.

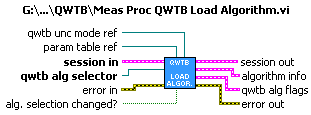


Figure 0‑18: Loader VI of the QWTB algorithm. The “session” is the measurement session of TWM, “qwtb alg selector” is the ref. to the ring with loaded and selected algorithm IDs. The “qwtb unc mode” and “param table” are references to the uncertainty mode selector ring and to the user parameters table.

Finally, when user confirms the calculation, the GUI will parse the eventual user parameters and stores them together with the selections made to a processing session. It does not store anything to the processing related file “qwtb.info” [7]! The processing setup is stored at the time of digitizing or before batch processing starts.

#### Algorithm execution routines

There are several processing related routines. First, the processing setup generated by the processing GUI must be stored to the processing info file “qwtb.info” in the measurement folder [7]. This is done by VI function “Meas Proc QWTB Write Algorithm Processing Header.vi”. The VI is called just once before the sequence of measurements as the processing setup is the same for all acquisitions.

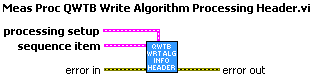


Figure 0‑19: Writer of QWTB processing setup. The "sequence item" contains the measurement root path. The "processing setup" is the setup obtained from processing setup GUI.

Next routine is the VI “Meas Process Record.vi”. This VI will initiate the QWTB algorithm execution by calling m-function “qwtb\_exec\_algorithm.m”. The VI does not transfer anything but measurement folder and record index to the Matlab. The rest of the configuration is obtained by the m-function from “qwtb.info”. Note the VI also does not query anything back from the Matlab (apart from eventual error). The results are stored to the measurement folder and this VI just notifies the other process of TWM that new result is available. The VI is equipped by the timeout capability, which allows to limit the processing time, but it will work only for GNU Octave. No way of terminating the Matlab Script Node was found yet. When the timeout ran out for GNU Octave, the calculation actually still runs. Just the VI returns and error. So the user may need to manually restart the GOLPI before next operations.

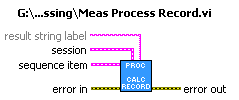


Figure 0‑20: Data processing executer. The “session” is measurement session of TWM, the “sequence item” contains measurement root path and index of current record. The “result string label” is not part of QWTB processing (it is used only for the raw m-code processing).

The “new result” notification VI “Meas Proc QWTB Notify Result Queue.vi” can be (is) called from anywhere from the TWM. It internally uses queue to which it stores the configuration flags that defines what the results viewer will do. Note it cannot be used before initiating the queue by VI “Meas Proc QWTB Initialize Result Queue.vi”!

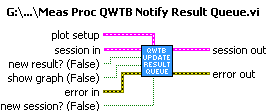


Figure 0‑21: New result notifier VI. The "session" is optional. The "plot setup" takes place only when the “show graph” is set. The “new session” must be set when new meas. folder is selected, the “new result” must be set when new record was processed or reprocessed, the “show graph” will display result as graph instead as a matrix. When no flag is set, the TWM will just refresh current result view.

#### Results viewer/interpreter

The calculated results are displayed by the results viewer process. It is a standalone process VI that is initiated from the main process of TWM whenever the “new result” queue contains the refresh notification. The asynchronous execution in another process was chosen because the process of retrieving the results and formatting the data take considerable time, so the synchronous execution in the GUI process would lockup the main panel GUI. However, the asynchronous execution just unloads the main panel, but the results viewer still uses the GOLPI to communicate with the Matlab. Therefore, when the refresh is initiated during the measurement, the algorithm execution routines (see above) and the results viewer shares the same Matlab instance, so only one of those two can operate at the time. This will result in delays in the processing or results viewing when the operations take longer time. However, the solution is in fact mostly effective, because the results viewing happens when the digitizers are acquiring new record, so there are no collisions.

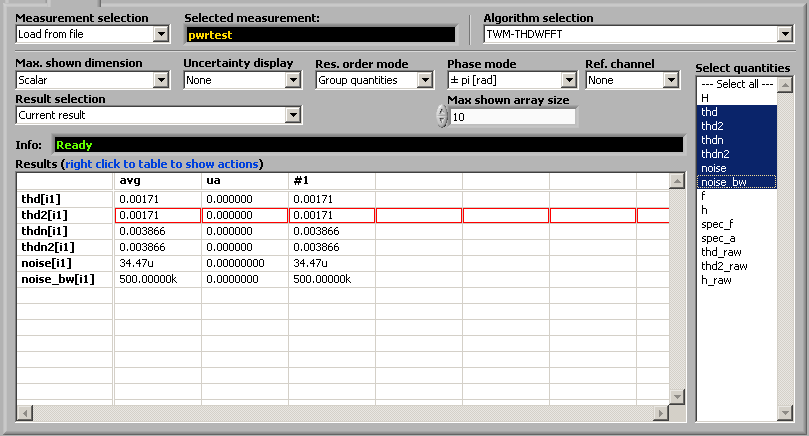


Figure 0‑22: Results viewer panel.

The periodic checking of the “new result” and eventual execution of the main viewer process is done in the timeout event of TWM main panel GUI by VI “Meas Proc QWTB Update Result View.vi”. The VI checks execution state of the previous call of the results viewer process (ref. to the process is entered from the local variable “qwtb view VI ref”). If the process is finished, it will check the state of the “new result” notifier and eventually initiate the results viewer process. The reference to the new process is returned and stored back to the “qwtb view VI ref”. The VI may or may not return the “qwtb view” depending on the execution state of the process itself. When it just returned, the TWM stores the viewer session “qwtb view” back to the local variable. Note the “qwtb view VI ref” must not be lost, otherwise there will be memory leakage of unclosed references!

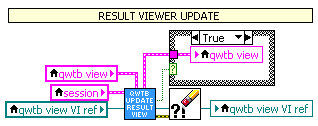


Figure 0‑23: TWM results viewer process executer (located in main panel GUI event structure in the timeout event). Note the “qwtb view VI ref” must be preserved in the local variable otherwise there would be memory leakage! This VI may or may not return the “qwtb view” depending on the state of execution of the results viewer process.

The results viewer process itself is VI “Meas Proc QWTB Update Result View Process.vi”. The VI accepts references to all the results viewing GUI controls, e.g. the selectors of the algorithm, quantities, etc. Is starts by obtaining the information about the selected measurement, i.e. it queries list of available results for current (or selected) correction folder. This is done by call of m-function “qwtb\_get\_results\_info.m”. It fills in the GUI controls: list of processed algorithms; list of quantities for selected algorithm; list of channels/phases. Next, the VI takes the result view selector value and uses them as parameters for m-function call “qwtb\_get\_results.m” for matrix display or “qwtb\_plot\_results.m” for graph display. When successful, the VI will update the results matrix in the main panel.



Figure 0‑24: Results viewer VI. Note the "session in" and "QWTB result session out" were intededly converted to "variable" data type, which is much easier to handle when using assycnhronou function calls.

Eventual export of the results table to the Excel sheet is performed in the main panel of TWM in the event structure. The exporting and other functions are available in right click popup menu of the results matrix.

TODO: condensed report export.

#### Batch processing GUI

TWM enables either runtime processing of the records or a batch processing. The batch processing is performed in the panel “Meas Batch Proc QWTB panel.vi”, which is shown in Figure 0‑26. The logic flow of the batch processor is shown in Figure 0‑25. The panel allows user to select measurement session and select the particular records to be processed. User must also set the configuration of the processing. This configuration will invoke the “Processing configuration GUI” as was described above. When the configuration is confirmed, the new calculation setup is store to the “qwtb.info” by VI call “Meas Proc QWTB Write Algorithm Processing Header.vi”. Next use may initiate the processing, which will call the “Meas Process Record.vi” for each record. The panel has no other relevant sub-VIs.

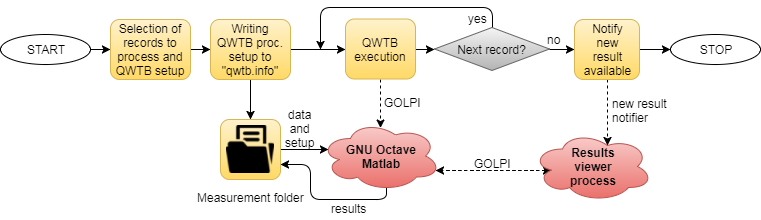


Figure 0‑25: Logical flow of the TWM’s batch processing of the previously recorder measurements.

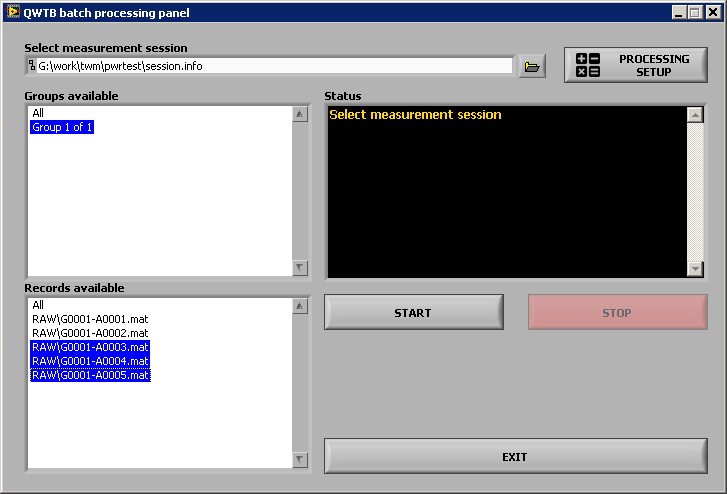


Figure 0‑26: Batch processing panel.

### Processing module – Matlab component

The Matlab component of the processing module is standalone set of m-functions. They are executed from TWM via the GOLPI interface [4]. From typical user point of view the only four functions are to be called directly:

1. “qwtb\_exec\_algorithm.m” for execution of the processing on the measurement session.
2. “qwtb\_get\_results\_info.m” to get information about available results in the session.
3. “qwtb\_get\_results.m” to load and format results for displaying in matrix form.
4. “qwtb\_plot\_result.m” to load and display results as a graph.

The rest of the m-functions are either sub-functions of abovementioned or special functions that are rarely used directly. Only the top level functions will be described in following sections.

#### qwtb\_exec\_algorithm.m

Function “qwtb\_exec\_algorithm.m” is top level function for execution of the algorithm on the TWM data. This is the function to be called when new processing is to be initiated. It performs all steps needed for the processing:

1. Load record(s) and correction files from the measurement folder using function “tpq\_load\_record.m”.
2. Loads processing setup from “qwtb.info” [7] from measurement session.
3. Executes the selected algorithm on each channel or phase of the measurement session.
4. Stores the results to the measurement folder using function “qwtb\_store\_results.m”.
5. Updates content of the “results.info” [7] so it contains information about the newly processed algorithm.

The details on the function call are shown in Table 0‑1.

Table 0‑1: Processing module m-function “qwtb\_exec\_algorithm.m” – execution of the TWM algorithm on the TWM data.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| qwtb\_exec\_algorithm(meas\_file, calc\_unc, is\_last\_avg, avg\_id, group\_id) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| meas\_file | Char string | Full path to the measurement session header INFO file “…/session.info”. |
| calc\_unc | Char string | Override uncertainty calculation mode from “qwtb.info”. Allowable values: ‘’ (default), ‘none’, ‘guf’ or ‘mcm’ (see QWTB documentation [3]). |
| is\_last\_avg | Bool | Flag that should be set to confirm all records for the calculation are in available. If not set and algorithm requires multiple records at once, the function will do nothing (no error). |
| avg\_id | Integer | Optional index (1, 2, 3, …) of the repetition cycle of the measurement session to process. The function processes the last available record if value is zero. |
| group\_id | Integer | Optional index (1, 2, 3, …) of the group index of the measurement session. Zero value selects the last available group. |
| ***Returns values:*** | | |
| None. Function can only generate errors. | | |

#### qwtb\_get\_results\_info.m

This m-function will retrieve selected information on the available results in the selected measurement folder. This function is mainly intended for the interface to the TWM, so the return values are formatted in a way that is easily readable by GOLPI [4]. The vectors of strings and matrices are returned as a CSV style strings. E.g. the Matlab table “A = [‘code’,’55’;’msg’,’Hallo’]” will be returned as a string: “code\t55\nmsg\tHallo”. Note the column and row separators may vary. The function prototype and return values are listed in the Table 0‑2.

Table 0‑2: Processing module m-function “qwtb\_get\_results\_info.m” – query of the TWM results information.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| [res\_files, res\_exist, alg\_list, chn\_list, var\_names]  = qwtb\_get\_results\_info(meas\_root, alg\_id) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| meas\_root | Char string | Path to the measurement session folder (no session file name!). |
| alg\_id | Char string | QWTB algorithm ID string, e.g. “TWM-PWRTDI”. Leave empty to load the last available results from the “results.info” [7]. |
| ***Returns values:*** | | |
| **Name** | **Data type** | **Description** |
| res\_files | Char string | CSV string with the list of available results files for the given measurement folder and algorithm. The list is separated by ‘\t’. |
| res\_exist | Bool | Non-zero of the result(s) exists. |
| alg\_list | Char string | CSV string with the list of algorithm IDs that were processed in the measurement folder. The list is separated by ‘\t’. |
| chn\_list | Char string | CSV string with the list of available channel/phase names for the selected algorithm. The list is 2D matrix with ‘,’ as column separator and ‘;’ as a row separator. Each row contains one channel name (e.g. “u1,i1” for single input algorithms) or single phase name (e.g. “L1” for dual input algs.). |
| var\_names | Char string | CSV string with the list of available result quantities for given algorithm. The list is separated by ‘\t’. |
| ***Errors:*** | | |
| The function won’t return error when “meas\_root” contains no results (yet). It will return error only if the desired “alg\_id” is not found or the data in the measurement folder are inconsistent. | | |

#### qwtb\_get\_results.m

This m-function will retrieve and format the selected results data in to a text matrix. The function performs following operations:

1. Selects the result from the measurement folder and loads its data using function “qwtb\_load\_results.m”. The function also enables selection of the quantities to be loaded and optional averaging of multiple results.
2. Formats the quantities to a matrix form in one of the supported view modes (scalars, vectors, matrices).

The function plots the data in two ways. When all quantities are scalar and “cfg.max\_dim” is set to scalar, it will show the quantities development in time, i.e. their values for each repetition cycle of the measurement. If the “cfg.max\_dim” is set to vector mode, the vector(s) of the quantities will be shown horizontally. Note the function limits maximum allowable number of element to be displayed by option “cfg.max\_array”, because displaying 1000 values would be extremely slow. If the amount of data exceeds the “cfg.max\_array”, the quantity in the table “txt” will contain “only graph” string instead of the data to indicate user may display the long vector as plot by function “qwtb\_plot\_result.m”.

Note this function is primarily intended for linking to the TWM tool, so it returns the text matrices in a CSV string format which is easier to handle by LV and GOLPI interface [4]. E.g. the Matlab 2D cell-array “A = {‘code’,’55’;’msg’,’Hallo’}” will be returned as a string: “code\t55\nmsg\tHallo”. Note the column and row separators may vary for various returned variables. The function prototype and return values are listed in the Table 0‑3.

Table 0‑3: Processing module m-function “qwtb\_get\_results.m” – query of the TWM results as a text matrix.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| [txt, desc, var\_names, chn\_index]  = qwtb\_get\_results(meas\_root, res\_id, alg\_id, cfg, var\_list) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| meas\_root | Char string | Path to the measurement session folder (no session file name!). |
| res\_id | Integer | Index of result(s) to load. Use -1 to load last available result, >1 to select particular result or 0 to average all results. |
| alg\_id | Char string | QWTB algorithm ID string, e.g. “TWM-PWRTDI”. Leave empty to load the last available results from the “results.info” [7]. |
| cfg | Structure | Configuration structure for the results formatting. All elements are optional.  cfg.max\_dim = maximum shown dimension of quantity {0: scalars, 1: vectors, 2: matrices}  cfg.max\_array = Maximum size of vector to be shown in the matrix  cfg.unc\_mode = Uncertainty display mode {0: none, 1: val±unc, 2: alternating rows val,unc,val,unc,…}  cfg.group\_mode = Grouping of the multichannel results {0: sort by channels, 1: sort by quantities}  cfg.phi\_mode = phase display mode {0: ±pi, 1: 0-2pi, 2: ±180°, 3: 0-360°}  cfg.phi\_ref\_chn = non-zero index of reference channel to use as a phase reference, use zero to disable interchannel phase display |
| var\_list | Cell array of char string | List of quantity names to load and display. Empty list will load all quantities. The list may contains only the names obtained by the “qwtb\_get\_results\_info.m” function. |
| ***Returns values:*** | | |
| **Name** | **Data type** | **Description** |
| txt | Char string | CSV string with 2D matrix of the formatted result values with ‘\t’ as column separator and ‘\n’ as row separator. First row contain table headers, first column contains quantity names with channel/phase indices. The rest of table are formatted quantity values. |
| desc | Char string | CSV string with 1D matrix of text descriptions of each data row of the “txt” matrix. The list is separated by ‘\t’. |
| var\_names | Char string | CSV string with 1D matrix of short quantity name for each data row of the “txt” matrix. The list is separated by ‘\t’. |
| chn\_index | 1D array of integers | Vector of channel/phase index for each row of “txt”. |
| ***Errors:*** | | |
| The function should throw an error only if the results data is inconsistent, which should not happen. | | |

#### qwtb\_plot\_result.m

This function is equivalent of the “qwtb\_get\_results.m”, except it will display the one selected quantity as a plot, instead of in the matrix. The plot can be made from a single record or average of records. If the scalar quantities are selected, the function will plot quantity value for each repetition cycle of the measurement. If the vector quantities are selected, the function plots the vector. Matrix quantities are not supported. The detail on the function are shown in Table 0‑4.

Table 0‑4: Processing module m-function “qwtb\_plot\_result.m” – plotting selected quantity of the TWM results.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| [] = qwtb\_plot\_result(meas\_root, res\_id, alg\_id, chn\_id, cfg, var\_name, plot\_cfg) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| meas\_root | Char string | Path to the measurement session folder (no session file name!). |
| res\_id | Integer | Index of result(s) to load. Use -1 to load last available result, >0 to select particular result or 0 to average all results. |
| Alg\_id | Char string | QWTB algorithm ID string, e.g. “TWM-PWRTDI”. Leave empty to load the last available results from the “results.info” [7]. |
| chn\_id | Integer | Index of channel/phase to plot. Use zero to plot all channels/phases in one plot. |
| Cfg | Structure | Configuration structure for the results formatting. All elements are optional.  cfg.max\_dim = maximum shown dimension of quantity {0: scalars, 1: vectors, 2: matrices}  cfg.phi\_mode = phase display mode {0: ±pi, 1: 0-2pi, 2: ±180°, 3: 0-360°}  cfg.phi\_ref\_chn = non-zero index of reference channel to use as a phase reference, use zero to disable interchannel phase display |
| var\_name | Char string | Name of the quantity to plot. The name must exist in the results file. |
| plot\_cfg | Structure | plot\_cfg.xlog = Non-zero to enable x-axis log scale.  Plot\_cfg.ylog = Non-zero to enable y-axis log scale.  Plot\_cfg.box = Show plot box (see Matlab “plot” doc).  Plot\_cfg.grid = Show plot grid (see Matlab “plot” doc).  Plot\_cfg.legend = Legend display position. (see Matlab “plot” doc for position names). Leave empty to disable plot. |
| ***Returns values:*** | | |
| None. | | |
| ***Errors:*** | | |
| The function should throw an error only if the results data is inconsistent, which should not happen. | | |

#### qwtb\_load\_algorithms.m

This function is used by the TWM to get list of available QWTB algorithms compatible with TWM. The list is created from the “qwtb\_list.info” file. The function is intended for the linking to the TWM via the GOLPI, so the returned arrays of strings were converted to the CSV strings, which are easier to handle. E.g. the Matlab 2D cell-array “A = {‘code’,’55’;’msg’,’Hallo’}” will be returned as a string: “code\t55\nmsg\tHallo”. Details on the function are shown in Table 0‑5.

Table 0‑5: Processing module m-function “qwtb\_load\_algorithms.m” – obtains list of the available TWM algorithms.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| [ids, names] = qwtb\_load\_algorithms(list\_file) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| list\_file | Char string | Path to the INFO-strings [2] file “qwtb\_list.info” with the list of supported TWM algorithms and their configuration. |
| ***Returns values:*** | | |
| **Name** | **Data type** | **Description** |
| ids | Char string | CSV string of the QWTB algorithm ID strings. The list is separated by the ‘\t’. |
| names | Char string | CSV string of the QWTB algorithm names. The list is separated by the ‘\t’. |
| ***Errors:*** | | |
| The function may throw an error if inconsistent data are in the “qwtb\_list.info” or in the QWTB algorithm wrappers. | | |

#### qwtb\_load\_algorithm.m

This function is used by TWM to obtain information about the selected algorithm. It is intended for the linking to the TWM via the GOLPI, so the returned arrays of strings were converted to the CSV strings, which are easier to handle. E.g. the Matlab 2D cell-array “A = {‘code’,’55’;’msg’,’Hallo’}” will be returned as a string: “code\t55\nmsg\tHallo”. Details are shown in Table 0‑6.

Table 0‑6: Processing module m-function “qwtb\_load\_algorithm.m” – obtains algorithm parameters.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| [alginfo, ptab, input\_params, is\_multi\_inp, is\_diff, has\_ui, unc\_guf, unc\_mcm] = qwtb\_load\_algorithm(alg\_id) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| alg\_id | Char string | QWTB algorithm ID [3] of the algorithm to select. |
| ***Returns values:*** | | |
| **Name** | **Data type** | **Description** |
| alginfo | Char string | QWTB algorithm info as returned by QWTB [3]. |
| Ptab | Char string | CSV string with the 1D matrix of algorithm parameters. Rows are separated by the ‘\n’. |
| input\_params | Char string | CSV string with the 1D matrix of algorithm’s parameter names. Rows are separated by the ‘\n’. |
| is\_multi\_inp | Bool | Flag that indicates the algorithm accepts multiple records at once. |
| Is\_diff | Bool | Flag indicates the algorithm supports differential inputs. |
| has\_ui | Bool | Flag that indicates the algorithm requires two inputs (voltage and current). |
| unc\_guf | Bool | Flag that indicates the algorithm supports GUF uncertainty calculation. |
| unc\_mcm | Bool | Flag that indicates the algorithm supports Monte Carlo uncertainty. |
| ***Errors:*** | | |
| The function may throw an error if inconsistent data in the QWTB algorithm wrappers. | | |

#### tpq\_load\_record.m

This function is main loader function for the TWM measurement. It performs following steps:

1. Loads common information from the measurement session INFO file [7].
2. Loads the selected records of from the measurement folder.
3. Loads transducer corrections by function “correction\_load\_transducer.m”.
4. Loads digitizer corrections by function “correction\_load\_digitizer.m”.

The function is capable to load one or more records from given measurement group. It returns one data structure with all data and parsed corrections. The details on the function call are shown in Table 0‑7.

Table 0‑7: Processing module m-function “tpq\_load\_record.m” – load TWM measurement.

|  |  |  |
| --- | --- | --- |
| ***Function prototype:*** | | |
| function [data] = tpq\_load\_record(header, group\_id, repetition\_id, data\_ofs, data\_lim) | | |
| ***Parameters:*** | | |
| **Name** | **Data type** | **Description** |
| header | Char string | Path to the measurement session file “./session.info” [7]. |
| group\_id | Integer | Default value -1 will select last available group. Value >0 will select particular measurement group. |
| repetition\_id | Integer | Index of the repetition cycle (record) to load. Value -1 will load the last record, value 0 will load all records, value >0 will load particular record. |
| data\_ofs | Integer | Optional sample-offset of the loader. Non-zero value means the loader will skip “data\_ofs” samples of the record(s). Note it will adapt the timestamp value(s) accordingly, so the timestamp(s) still applies to the first sample. |
| data\_lim | Integer | Non-zero value limits the amount of loaded samples per channel to “data\_lim” samples. |
| ***Returns values:*** | | |
| **Name** | **Data type** | **Description** |
| data | Structure | Structure containing the loaded sample data and corrections, see Table 0‑8 for details. |
| ***Errors:*** | | |
| The function will throw an error if the record selection is invalid, or if there are inconsistent data anywhere in the measurement session or the correction data. | | |

Table 0‑8: Processing module m-function “tpq\_load\_record.m” – output structure data.

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description** |
| group\_count | Integer | Total measurement groups count in the measurement session. |
| repetitions\_count | Integer | Total repetition cycles (records) in the selected group. |
| channels\_count | Integer | Digitizer channels in the measurement session. |
| is\_temperature | Bool | Temperature measurement available. |
| sample\_count | Integer | Samples count per channel in the loaded record(s). |
| y | Double | Sample data, one row per channel. If multiple records are loaded, the records are merged horizontally:  chn1,chn2, chn1,chn2, … |
| timestamp | Double | Relative timestamp(s) in seconds for the first sample(s) of each record in the “y”. |
| Ts | Double | Sampling period in seconds. |
| corr | Structure | Data structure containing the loaded corrections:  corr.phase\_idx = phase index for each digitizer channel which is used to define which channels belongs together for multi-input algorithms  corr.dig = digitizer corrections structure  corr.tran{} = cell array of the transducer corrections, one for each transducer |