# A2.4.5 - TWM structure

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>

## Abbreviations

LV – LabVEIW

CVI – LabWindows CVI

EOS – End of string

DWORD – unsigned 32bit variable

INT16 – signed 16bit integer

INT32 – signed 32bit integer

Float32 – 32-bit real number

BYTE – unsigned 8bit variable

HDD – Hard drive

TWM – The LV program developed in scope of TracePQM project

GUI – Graphical User Interface

HW – HardWare

QWTB – Q-Wave toolbox

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. Matlab/GNU Octave module 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).

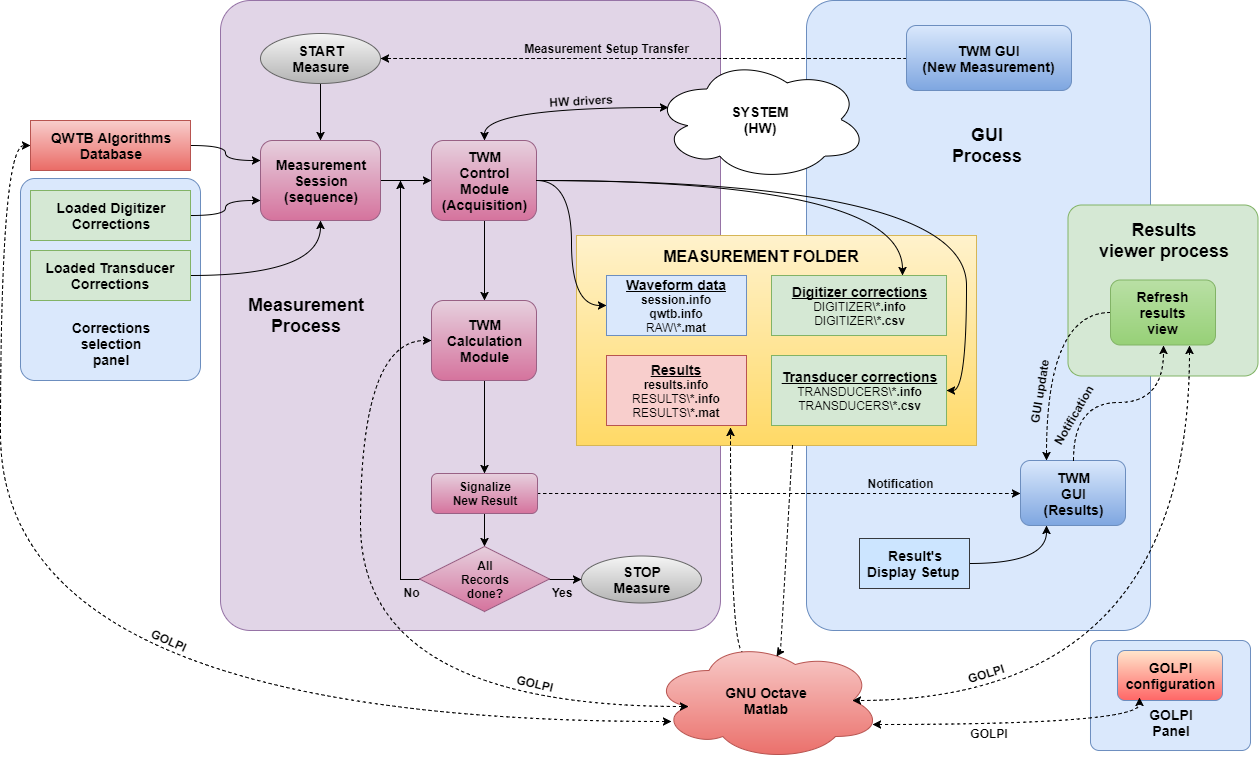


Figure 0‑1: TWM structure.

The modules communicate on runtime via the GOLPI interface [4] and via files in the measurement folder. 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 actual 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 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 (supercomputer).
5. The Processing module is identical for LabVIEW and CVI version of the tool and the data are interchangeable.

The control module is split into three separate processes that runs 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.

## 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 so it is not restricted to LV. 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 Matlab Script nodes. The nodes are hidden in the GOLPI so from outside there is no difference between use for Matlab and GNU Octave and there should be no difference apart from the performance, which is strongly dependent on version. The only difference may be in some algorithms, where Matlab and GNU Octave implementation differs (see algorithms documentation).

## Control and data acquisition module

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

### Control module

Description to be done…

### Acquisition module

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

Description to be done…

#### Modular driver design

The modular driver concept was developed in A2.1.2, A2.2.3. The key idea is the Acquisition module does not access the drivers of the particular instruments directly, because each digitizer requires completely different approach. Instead a command translation layer is put in between. This layer is called virtual digitizer. All HW specific function calls of each digitizer are translated to universal format and merged into the few basic VI functions which are, for a user, identic for any digitizer no matter how different the HW control is. The main set of functions is shown in Figure 0‑2 in a green colour. The diagram also shows a logic flow of the acquisition process.

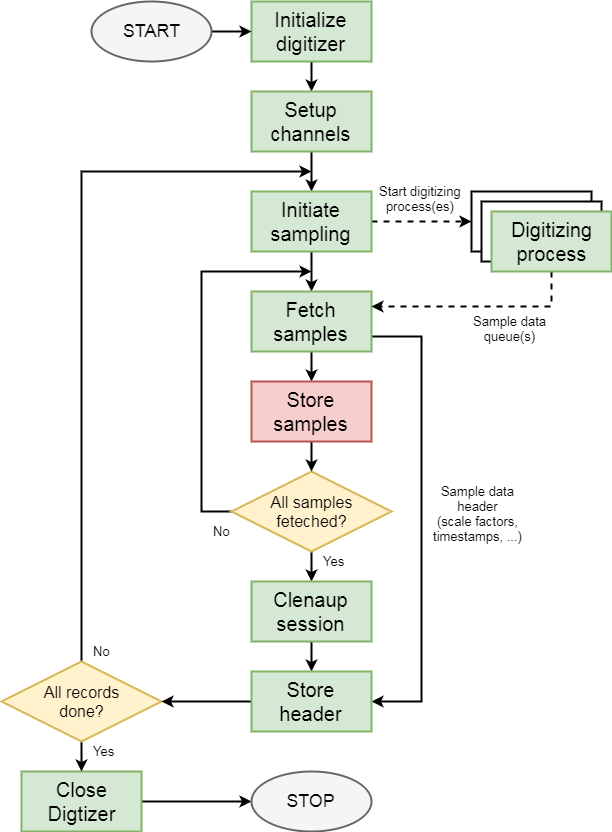


Figure 0‑2: Virtual digitizer driver structure and function.

To fix/revwrite/remove:

The structure cannot be easily simplified because one of the requirements of the project was the long term streaming of sample data. It is not possible to digitize more sample data than can fit into computer memory (further limited by 32bit implementation to some 2 GiB), therefore the structure must allow runtime streaming of the sample data to a disk. However, it would be very impractical to integrate the “Store samples” function inside the virtual digitizer, because each digitizer would have another implementation of this function and whenever the format is to be changed due to future extensions, it would be necessary to fix it in every driver. Therefore a more complex but flexible method was developed for some of the drivers as shown in Figure 0‑2. The samples are acquired in parallel running process(es) and the user just fetches and stores the data in a loop. This also allowed termination of the sampling at any point even for HW that does not allow this easily, because the sampling process can be “killed” by a STOP button at any point. In fact the solution with parallel processes seems to be the only method for the 3548A streaming drivers.

To integrate a new driver, one must implement the virtual driver functions so they follow the generalized driver functions shown below. They don’t have to follow exactly, they may omit some parameters or whole functions (marked “optional”), but they have to “fit” into the generalized functions because changes in the TWM structure are not possible without loss of compatibility.

#### Initialize driver (optional)

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

#### Enumerate devices (optional)

This optional function is called on the initialization of the TWM and 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.

#### Initialize digitizer (required)

It is first function to be called 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.

#### Setup channels (required)

This function 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. After this function call the digitizer shall be in ready to start acquisition.

#### Initiate sampling (required)

This function should immediately initiate the sampling (arm the virtual digitizer). And return. The actual operation depends on the “Fetch samples” function (see below).

#### Fetch samples (required)

Purpose of this function is to obtain the acquired data from the virtual digitizer channels. There are three 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 is TWM cannot query state of the sampling and the termination by STOP command is harder to implement. This is typical way the most of instrument drivers are made.
2. Asynchronous function that just checks weather the sampling is done. When not, it will just return status. 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 does not allow asynchronous operation.
3. Complex implementation shown in the Figure 0‑2 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”).

#### 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 should be called every time to cleanup before new acquisition or before termination.

#### Close digitizer (required)

This function shall 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!

#### 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).

#### Get Digitizer Capabilities (required)

This function returns capabilities of the selected digitizer. 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”! TWM decides by itself when to call these two functions so the time consuming querying is not perform when it is not needed. It is used to limit the GUI entries and disable unsupported features. The driver should just query the information from the session and return desired capabilities. There is a cluster of standard capabilities, such as range of sampling rate, range of samples count, aperture range, etc.

#### 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. There are returned as an array of text strings pairs, i.e. [“name”, “value”]. These are not used by the TWM, they are just stored to the measurement session header. Note the “value” must be formatted with decimal dot.

#### GUI Get Info (recommended)

This function takes digitizer session and returns brief description of the digitizer which is displayed in the digitizer panel. It may contain e.g. trigger connection notes (3458A mode).

#### 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.