Report A2.1.4: Concept of Interfacing LabWindows/CVI to Matlab.

Objective

This report is related with activity A2.1.4: “The concept for the interface between the data processing module and the control and data acquisition module”, of TracePQM-15RPT04, and only describes the concept for interfacing LabWindows/CVI to MATLAB tool.

The Labview to MATLAB /Octave interface was already developed with GOLPI library [1] and lv\_process, the low level library to access hardware via pipes. The required interface for LabWindows/CVI to MATLAB has been developed using MATLAB Engine API. [2] This API requires also MATLAB installed, not only Runtime Engine, to use properly, related to documentation given form MATLAB internet site.

Structure

In order to use MATLAB functions/scripts from C or LabWindows/CVI environment, MATLAB tool must be installed in the system. The operating system, hardware and all related software should use the same bit number, 32-Bit or 64-Bit.

The last version of TWM tool should be downloaded from source site.[3] The TWM tool uses MATLAB and QWTB tool, the Quantum Wave ToolBox. [4]The other related libraries, niscope, niTClk, should be exist and accessible from user space in order to run virtual digitizers used in TWM software.

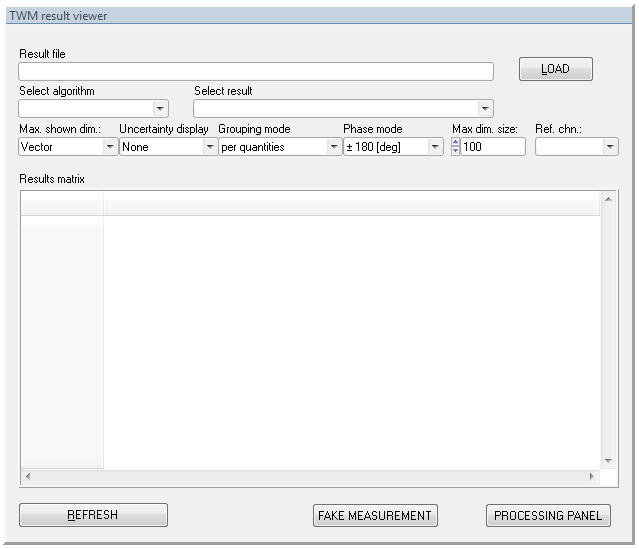


Figure 1 GUI window of the module

The CVI tool, the second version is shown can be downloaded from TracePQM web site, both source codes and compiled as a standalone application. [5] The CVI tool uses QWTB tool and its scripts like TWM application and performs the desired calculations. When it runs, it looks like the picture shown in Figure 1, similar to QWTB calculation section of TWM software.

The results obtained from digitizers are stored as result files with required format and many calculations can be performed with CVI tool being used MATLAB scripts background.

LabWindows/CVI to MATLAB interfacing is done with using dedicated header files, library and MATLAB Engine. The CVI project/program must include these header files and use the functions of libraries. The libeng.lib and libmx.lib files must be included in project tree and the system-wide path of these files must be added the operating system paths in order to run MATLAB engine calls.

Implementation

The name of CVI module is “Matlab Module” and name of related c file is Matlab Module.c. The name of GUI (Graphic User Interface) of the module is Matlab Module.uir. The screenshot of the window is shown in Figure 1.

When the application runs, it first initiates GUI and calls the main procedure, named main, located in Matlab Module.c.

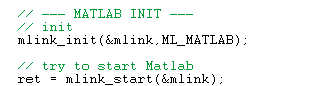


Figure 2 MATLAB engine initialization and starting in main procedure.

This procedure also initiates the MATLAB session calling the mlink\_init, declared in mlink.c source file and starts the MATLAB engine at the background calling mlink\_start, declared in the same source file, Figure 2, Figure 3, respectively.

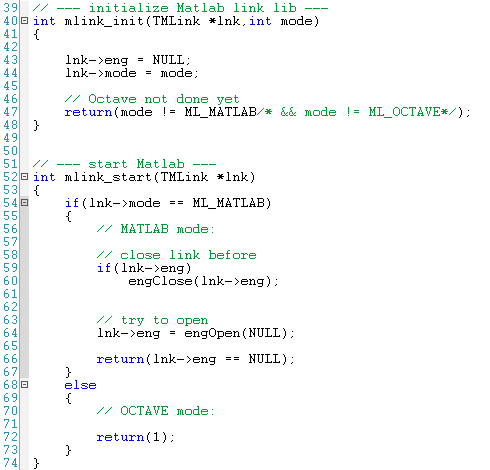


Figure 3 Declaring MATLAB engine start procedure in mlink.c source file.

The TMLink parameter is a C struct and declared in mlink.h header files. ML\_MATLAB and ML\_OCTAVE constants are also declared in the same header file, shown in Figure 4.

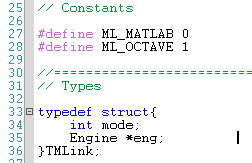


Figure 4 TMLink struct.

The MATLAB engine opens with engOpen command and returns a pointer for handle of MATLAB engine. The init parameter passed to function must be NULL for Windows operating systems, declared in [1].

The first interaction of MATLAB environment is completed with the command sequence. Now, we have an engine handle to be used in LabWindows/CVI environment.

The main procedure runs the twm\_init command, declared in twm\_matlab.c, with the parameter which returned from the mlink\_start procedure, shown in Figure 5, if there are no errors from previous operations.

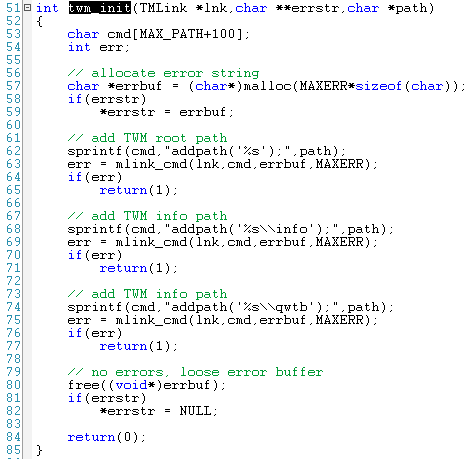


Figure 5 Declaration of twm\_init procedure.

When the twm\_init runs, the MATLAB is ready to run with the desired working paths. This command calls the mlink\_cmd command, declared in mlink.c source file, shown in Figure 6. This command calls the engine API procedures, engEvalString and engOutputBuffer including correct paths and command string to be evaluated by MATLAB. The engEvalString command evaluates the expression contained in command string for the MATLAB engine session, lnk, previously started by engOpen. The engOutputBuffer command defines a character buffer for engEvalString to return any output that ordinarily appears on the screen. This buffer will be filled with output values of MATLAB and used by main program.

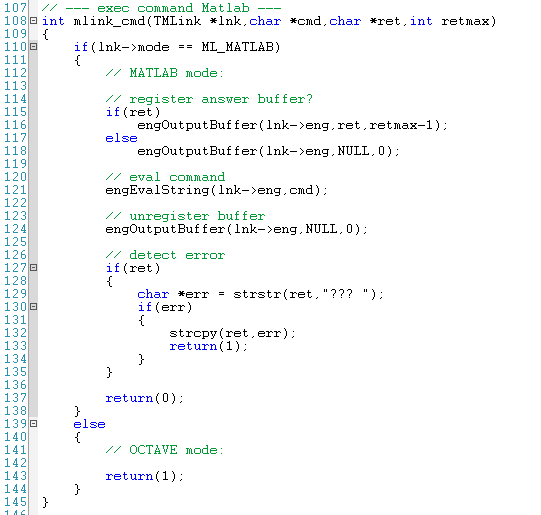


Figure 6 mlink\_cmd command structure.

After the main program initiates, the user must select the proper info file by pressing LOAD button on the top-right side of main window. The open-file dialog of operation system appears and waits an info file to select when this button is pressed, shown in Figure 7.

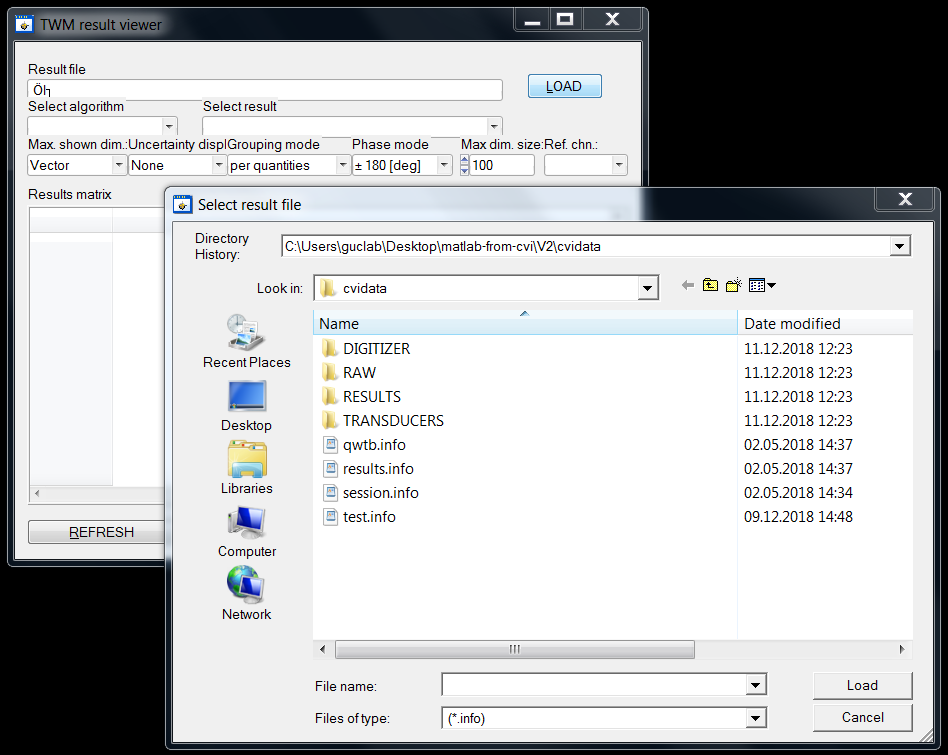


Figure 7 Loading the info file.

The session.info file can be selected as an example. The format of info file, including all required digitizer, transducer corrections, offsets, data formats, raw data etc, was described detailed in activity A2.3.1.

When the selected file loaded, it should be clicked the REFRESH button at the left-bottom of window. With activating the REFRESH button, the dedicated CALLBACK procedure of main GUI evaluated. The CALLBACK function performs a list of command sequence and determines the required evaluation/calculation parameters such as algorithm type, the paths for info, results, corrections, assigns the related variables and calls the twm\_get\_result\_info, twm\_get\_results\_data, twm\_get\_alg\_list, twm\_get\_alg\_info commands, declared in twm\_matlab.c source file with passing the required parameter, the pointer to handle that represents MATLAB engine, to perform various operations on the measurement dates.

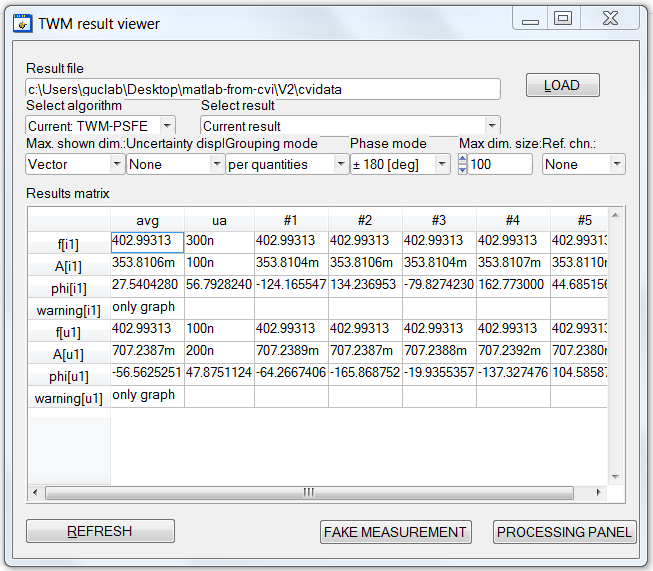


Figure 8 The actual window.

Conclusions

The basic interaction of LabWindows/CVI to MATLAB tool is briefly explained. The only MATLAB like calculation tool is MATLAB, so the OCTAVE part is not implemented by code developer at the moment. With this successful communication scheme of these two environments, the well known algorithms written in MATLAB have been used efficiently without re-written completely for the other languages.

References

[1] https://github.com/KaeroDot/GOLPI

[2] <https://uk.mathworks.com/help/matlab/calling-matlab-engine-from-c-programs-1.html>

[3] <https://github.com/smaslan/TWM/blob/master/builds/TWM.zip>

[4] <https://qwtb.github.io/qwtb/>

[5] <http://tracepqm.cmi.cz/database/files/wp2/task%202.3/20Matlab%20Module%20v2.zip>

Appendix

## The Matlab Module.c source file:

//==============================================================================

//

// Title: Matlab Module

// Purpose: A short description of the application.

//

// Created on: 2.3.2014 at 16:16:57 by .

// Copyright: . All Rights Reserved.

//

//==============================================================================

//==============================================================================

// Include files

#include <ansi\_c.h>

#include <windows.h>

#include <Shlwapi.h>

#include <cvirte.h>

#include <userint.h>

#include "Matlab Module.h"

#include "toolbox.h"

#include "qwtb\_alg\_select.h"

#include "mlink.h"

#include "twm\_matlab.h"

#include "utils.h"

#include "matlab\_globals.h"

//==============================================================================

// Constants

#define INIFILE "config.ini"

//==============================================================================

// Types

//==============================================================================

// Static global variables

static int panelHandle = 0;

//==============================================================================

// Static functions

//==============================================================================

// Global variables

// Matlab link handle

TMLink mlink;

// config.ini full path

char ini[MAX\_PATHNAME\_LEN];

// last result path

char resfld[MAX\_PATH];

//==============================================================================

// Global functions

/// HIFN The main entry-point function.

int main (int argc, char \*argv[])

{

int error = 0;

int ret;

// get app directory

char appdir[MAX\_PATHNAME\_LEN];

GetProjectDir(appdir);

// build ini path

strcpy(ini,appdir);

strcat(ini,"\\config.ini");

// load TWM function path from INI file

char twm\_path[MAX\_PATH];

ret = GetPrivateProfileString("PATH","twm\_octave\_folder","",twm\_path,MAX\_PATH,ini);

if(!ret)

{

MessageBoxA(NULL,"Missing INI file or the [PATH],twm\_octave\_folder value!","Error",0);

goto Error;

}

// load last result folder path from INI file

ret = GetPrivateProfileString("PATH","last\_result\_path","",resfld,MAX\_PATH,ini);

// --- MATLAB INIT ---

// init

mlink\_init(&mlink,ML\_MATLAB);

// try to start Matlab

ret = mlink\_start(&mlink);

if(ret)

{

MessageBoxA(NULL,"Cannot start Matlab!","Error",0);

}

else

{

// initialize TWM link

char \*errstr;

ret = twm\_init(&mlink,&errstr,twm\_path);

// failed?

if(ret)

{

MessageBoxA(NULL,errstr,"Matlab error",0);

free((void\*)errstr);

}

else

{

/\* initialize and load resources \*/

nullChk (InitCVIRTE (0, argv, 0));

errChk (panelHandle = LoadPanel (0, "Matlab Module.uir", PANEL));

/\* display the panel and run the user interface \*/

errChk (DisplayPanel (panelHandle));

errChk (RunUserInterface ());

}

}

Error:

/\* clean up \*/

if (panelHandle > 0)

DiscardPanel (panelHandle);

return 0;

}

//==============================================================================

// UI callback function prototypes

/// HIFN Exit when the user dismisses the panel.

int CVICALLBACK panelCB (int panel, int event, void \*callbackData,

int eventData1, int eventData2)

{

if (event == EVENT\_CLOSE)

{

QuitUserInterface (0);

// try to close Matlab

mlink\_close(&mlink,1);

// store last path

WritePrivateProfileString("PATH","last\_result\_path",resfld,ini);

}

return 0;

}

//------------------------------------------------------------------------------

// Refresh result view

//

int CVICALLBACK btn\_cmd (int panel, int control, int event,

void \*callbackData, int eventData1, int eventData2)

{

switch (event)

{

case EVENT\_COMMIT:

char path[MAX\_PATH];

// default alg/res selection:

int alg\_id = -1;

int res\_id = -1;

int ref\_id = -1;

// results found?

int res\_exist = 0;

// get result path

GetCtrlVal(panel,PANEL\_PATH\_RES,(void\*)path);

// -- two pass assignement

// 1) fill result/alg selectors

// 2) load selected alg and result

for(int pass = 0;pass < 2;pass++)

{

// get last algorithm selection

// 0 - last used

// 1,2,... - alg. selection ID

GetCtrlIndex(panel,PANEL\_RING\_ALG,&alg\_id);

if(alg\_id < 0)

alg\_id = 0;

// get last result selection

// 0 - last measured

// 1 - average of all

// 2,3,... - result selection ID

GetCtrlIndex(panel,PANEL\_RING\_RES,&res\_id);

if(res\_id < 0)

res\_id = 0;

// get ref channel mode selection

// 0 - no ref

// 1,2,... - channel selection ID

GetCtrlIndex(panel,PANEL\_REFMODE,&ref\_id);

if(ref\_id < 0)

ref\_id = 0;

// alg. selection string

char alg\_id\_str[256];

strcpy(alg\_id\_str,"");

if(pass && alg\_id)

GetLabelFromIndex(panel,PANEL\_RING\_ALG,alg\_id,(void\*)alg\_id\_str);

// read results

char \*res\_files;

char \*alg\_list;

char \*chn\_list;

char \*errstr;

int ret = twm\_get\_result\_info(&mlink,&errstr,path,alg\_id\_str,&res\_exist,&res\_files,&alg\_list,&chn\_list);

if(ret)

{

// failed

MessageBoxA(NULL,errstr,"Matlab error",0);

free((void\*)errstr);

return(1);

}

// clear result selectors

ClearListCtrl(panel,PANEL\_RING\_RES);

ClearListCtrl(panel,PANEL\_RING\_ALG);

ClearListCtrl(panel,PANEL\_REFMODE);

if(res\_exist)

{

// --- something found - fill the dialog

int rows,cols,cc;

char \*\*cells,\*csv;

// fill algorithms list

csv = (char\*)malloc((strlen(alg\_list)+100)\*sizeof(char));

strcpy(csv,"Current: ");

strcat(csv,alg\_list);

free((void\*)alg\_list);

cc = csv\_parse(csv,';','\t',&rows,&cols,NULL);

cells = (char\*\*)malloc(cc\*sizeof(char\*));

csv\_parse(csv,';','\t',&rows,&cols,cells);

InsertListItem(panel,PANEL\_RING\_ALG,0,cells[0],0);

for(int k=2;k<cc;k++)

InsertListItem(panel,PANEL\_RING\_ALG,k-1,cells[k],k-1);

free((void\*)cells);

free((void\*)csv);

if(alg\_id >= cc-1)

{

SetCtrlIndex(panel,PANEL\_RING\_ALG,0);

SetCtrlIndex(panel,PANEL\_RING\_RES,0);

}

else

{

SetCtrlIndex(panel,PANEL\_RING\_ALG,alg\_id);

}

// fill results list

csv = (char\*)malloc((strlen(res\_files)+100)\*sizeof(char));

strcpy(csv,"Current result\tAverage\t");

strcat(csv,res\_files);

free((void\*)res\_files);

cc = csv\_parse(csv,';','\t',&rows,&cols,NULL);

cells = (char\*\*)malloc(cc\*sizeof(char\*));

csv\_parse(csv,';','\t',&rows,&cols,cells);

for(int k=0;k<cc;k++)

InsertListItem(panel,PANEL\_RING\_RES,k,cells[k],k);

free((void\*)cells);

free((void\*)csv);

if(res\_id >= cc)

{

SetCtrlIndex(panel,PANEL\_RING\_RES,0);

res\_id = 0;

}

else

SetCtrlIndex(panel,PANEL\_RING\_RES,res\_id);

// fill results list

csv = (char\*)malloc((strlen(chn\_list)+100)\*sizeof(char));

strcpy(csv,"None;");

strcat(csv,chn\_list);

free((void\*)chn\_list);

cc = csv\_parse(csv,'\t',';',&rows,&cols,NULL);

cells = (char\*\*)malloc(cc\*sizeof(char\*));

csv\_parse(csv,'\t',';',&rows,&cols,cells);

for(int k=0;k<cc;k++)

InsertListItem(panel,PANEL\_REFMODE,k,cells[k],k);

free((void\*)cells);

free((void\*)csv);

if(ref\_id >= cc)

{

SetCtrlIndex(panel,PANEL\_REFMODE,0);

ref\_id = 0;

}

else

SetCtrlIndex(panel,PANEL\_REFMODE,ref\_id);

}

else

{

// failed

MessageBoxA(NULL,"Selected result is invalid or contains no calculated results.","Matlab error",0);

break;

}

}

if(res\_exist)

{

// -- load result data

// display config

TResCfg cfg;

GetCtrlVal(panel,PANEL\_MAXDIM,(void\*)&cfg.max\_dim);

GetCtrlVal(panel,PANEL\_MAXDIMSZ,(void\*)&cfg.max\_array);

GetCtrlVal(panel,PANEL\_GRPMODE,(void\*)&cfg.group\_mode);

GetCtrlVal(panel,PANEL\_UNCMODE,(void\*)&cfg.unc\_mode);

GetCtrlVal(panel,PANEL\_PHIMODE,(void\*)&cfg.phi\_mode);

GetCtrlVal(panel,PANEL\_REFMODE,(void\*)&cfg.phi\_ref\_chn);

// alg. selection string

char alg\_id\_str[256];

strcpy(alg\_id\_str,"");

if(alg\_id)

GetLabelFromIndex(panel,PANEL\_RING\_ALG,alg\_id,(void\*)alg\_id\_str);

// obtain result data from matlab

char \*csv = NULL;

char \*errstr;

int ret = twm\_get\_result\_data(&mlink,&errstr,path,res\_id-1,alg\_id\_str,&cfg,&csv);

if(ret)

{

// failed

MessageBoxA(NULL,errstr,"Matlab error",0);

free((void\*)errstr);

return(1);

}

int rows,cols;

int cc = csv\_parse(csv,'\t','\n',&rows,&cols,NULL);

char \*\*cells = (char\*\*)malloc(cc\*sizeof(char\*));

csv\_parse(csv,'\t','\n',&rows,&cols,cells);

// disable table refresh

SetCtrlAttribute(panel, PANEL\_TABLE, ATTR\_VISIBLE, 0);

// refresh table data size

DeleteTableColumns(panel,PANEL\_TABLE,1,-1);

DeleteTableRows(panel,PANEL\_TABLE,1,-1);

InsertTableRows(panel,PANEL\_TABLE,1,rows-1,VAL\_CELL\_STRING);

InsertTableColumns(panel,PANEL\_TABLE,1,cols-1,VAL\_CELL\_STRING);

// refresh headers

SetTableRowAttribute (panel, PANEL\_TABLE, -1, ATTR\_USE\_LABEL\_TEXT, 1);

for(int r = 1;r<rows;r++)

SetTableRowAttribute(panel,PANEL\_TABLE,r,ATTR\_LABEL\_TEXT,cells[r\*cols]);

SetTableColumnAttribute (panel, PANEL\_TABLE, -1, ATTR\_USE\_LABEL\_TEXT, 1);

for(int c = 1;c<cols;c++)

SetTableColumnAttribute(panel,PANEL\_TABLE,c,ATTR\_LABEL\_TEXT,cells[c]);

// refresh data

for(int r = 1;r<rows;r++)

SetTableCellRangeVals(panel,PANEL\_TABLE,MakeRect(r,1,1,cols-1),&cells[r\*cols+1],VAL\_ROW\_MAJOR);

// autoscale

for(int c = 1;c<cols;c++)

SetColumnWidthToWidestCellContents(panel,PANEL\_TABLE,c);

// enable table refresh

SetCtrlAttribute(panel, PANEL\_TABLE, ATTR\_VISIBLE, 1);

free((void\*)cells);

free((void\*)csv);

}

break;

}

return 0;

}

//------------------------------------------------------------------------------

// Select result file

//

int CVICALLBACK btn\_load (int panel, int control, int event,

void \*callbackData, int eventData1, int eventData2)

{

switch (event)

{

case EVENT\_COMMIT:

// open result file

char path[MAX\_PATH];

if(FileSelectPopup(resfld,"\*.info","\*.info","Select result file",VAL\_LOAD\_BUTTON,0,0,1,1,path)

== VAL\_EXISTING\_FILE\_SELECTED);

{

// extract folder path

char \*p = strrchr(path, '\\');

if(p) \*p = '\0';

// remember last result folder

strcpy(resfld,path);

// write file path to the box

SetCtrlVal(panel,PANEL\_PATH\_RES,path);

}

break;

}

return 0;

}

//------------------------------------------------------------------------------

// processing configuration panel

//

int CVICALLBACK btn\_proc\_cfg (int panel, int control, int event,

void \*callbackData, int eventData1, int eventData2)

{

switch (event)

{

case EVENT\_COMMIT:

int cfg\_panel = LoadPanel(0,"qwtb\_alg\_select.uir",PROCPANEL);

DisplayPanel(cfg\_panel);

break;

}

return 0;

}

//------------------------------------------------------------------------------

// fake generation of some measurement data

int CVICALLBACK btn\_fake\_proc (int panel, int control, int event,

void \*callbackData, int eventData1, int eventData2)

{

switch (event)

{

case EVENT\_COMMIT:

// open folder

char path[MAX\_PATH];

if(DirSelectPopup(resfld,"Select destination for fake measurement",1,1,path)

== VAL\_DIRECTORY\_SELECTED);

{

// crate some fake sample data

float smpl\_c1[1000];

float smpl\_c2[1000];

float smpl\_c3[1000];

for(int i = 0;i < 1000;i++)

{

smpl\_c1[i] = (float)(i%100);

smpl\_c2[i] = (float)(i%150);

smpl\_c3[i] = (float)(i%75);

}

// create record info structure

TTWMssnInf info;

memset((void\*)&info, 0, sizeof(info)); // always clear it before filling new stuff!!!

// now fill in the basic sampling informations

info.N = 1000; // samples count

info.fs = 100000; // sampling rate

info.chn\_count = 3; // digitizer channels count (not transducers!)

info.chn\_data\_type = TWMMATFMT\_SGL; // sample data in real32 format

// -- now build list of digitizre channel setups for up to TWMMAXTR = 6 channels

strncpy(info.chn\_idns[0],"channel 1",TWMMAXSTR); // create name of dig. channel 1

info.chn\_gains[0] = 1.0001; // gain factor of channel 1

info.chn\_offs[0] = 0.0003; // offset of channel 1

info.chn\_rng[0] = 10.0; // nominal range of channel 1

info.time\_stamps[0] = 1.2345; // relative timestamp of channel 1 (may be zero)

info.chn\_data[0] = smpl\_c1; // sample data pointer for channel 1

strncpy(info.chn\_idns[1],"channel 2",TWMMAXSTR); // create name of dig. channel 2

info.chn\_gains[1] = 1.0002; // gain factor of channel 2

info.chn\_offs[1] = 0.0004; // offset of channel 2

info.chn\_rng[1] = 10.0; // nominal range of channel 2

info.time\_stamps[1] = 1.2346; // relative timestamp of channel 2 (may be zero)

info.chn\_data[1] = smpl\_c2; // sample data pointer for channel 2

strncpy(info.chn\_idns[2],"channel 3",TWMMAXSTR); // create name of dig. channel 3

info.chn\_gains[2] = 1.0003; // gain factor of channel 3

info.chn\_offs[2] = 0.0005; // offset of channel 3

info.chn\_rng[2] = 10.0; // nominal range of channel 3

info.time\_stamps[2] = 1.2347; // relative timestamp of channel 3 (may be zero)

info.chn\_data[2] = smpl\_c3; // sample data pointer for channel 3

// next channels ...

// path to selected digitizer correction file

strcpy(info.dig\_corr,"c:\\TPQA\\corrections\\digitizer\\HP3458A\\HP3458\_v1.info");

// -- now define transducer and how their are connected to the digitizer channels:

// transducers count

info.tr\_count = 2;

// first transducer definition

strcpy(info.tr\_corr[0],"c:\\TPQA\\corrections\\transducers\\shunt\_1A1\\shunt\_1A1.info"); // path to correction file

info.tr\_phase[0] = 1; // phase index of transducer (use the same index for U-I pair for power, but different indices if measuring multi phse U or multiphase I)

info.tr\_map[0][0] = 2; // mapping to digitizer for differential connection [][2]-[][3], i.e. dig channels 2-3

info.tr\_map[0][1] = 3;

// second transducer definition

strcpy(info.tr\_corr[1],"c:\\TPQA\\corrections\\transducers\\rvd\_230V1\\shunt\_230V1.info");

info.tr\_phase[1] = 1;

info.tr\_map[0][0] = 1; // mapping to digitizer: example of single-ended connection to dig. channel 1

// next transducer definitions...

// copy QWTB setup from config panel (not nice solution! may be uninitalized!)

info.qwtb = cfg\_qwtb;

// generate measurement session

twm\_write\_session(path, &info);

}

break;

}

return 0;

}

## m\_link.c source file:

//==============================================================================

//

// Title: matlab\_module.c

// Purpose: A short description of the implementation.

//

// Created on: 2.3.2014 at 17:46:31 by .

// Copyright: . All Rights Reserved.

//

//==============================================================================

//==============================================================================

// Include files

#include <ansi\_c.h>

#include "mlink.h"

#include "engine.h"

#include "matrix.h"

//==============================================================================

// Constants

//==============================================================================

// Types

//==============================================================================

// Static global variables

//==============================================================================

// Static functions

//==============================================================================

// Global variables

//==============================================================================

// Global functions

// --- initialize Matlab link lib ---

int mlink\_init(TMLink \*lnk,int mode)

{

lnk->eng = NULL;

lnk->mode = mode;

// Octave not done yet

return(mode != ML\_MATLAB/\* && mode != ML\_OCTAVE\*/);

}

// --- start Matlab ---

int mlink\_start(TMLink \*lnk)

{

if(lnk->mode == ML\_MATLAB)

{

// MATLAB mode:

// close link before

if(lnk->eng)

engClose(lnk->eng);

// try to open

lnk->eng = engOpen(NULL);

return(lnk->eng == NULL);

}

else

{

// OCTAVE mode:

return(1);

}

}

// --- close Matlab ---

int mlink\_close(TMLink \*lnk,int close)

{

if(lnk->mode == ML\_MATLAB)

{

// MATLAB mode:

if(lnk->eng)

{

// evaluate "exit" command?

if(close)

{

engEvalString(lnk->eng,"exit");

}

// close engine

engClose(lnk->eng);

lnk->eng = NULL;

}

return(0);

}

else

{

// OCTAVE mode:

return(1);

}

}

// --- exec command Matlab ---

int mlink\_cmd(TMLink \*lnk,char \*cmd,char \*ret,int retmax)

{

if(lnk->mode == ML\_MATLAB)

{

// MATLAB mode:

// register answer buffer?

if(ret)

engOutputBuffer(lnk->eng,ret,retmax-1);

else

engOutputBuffer(lnk->eng,NULL,0);

// eval command

engEvalString(lnk->eng,cmd);

// unregister buffer

engOutputBuffer(lnk->eng,NULL,0);

// detect error

if(ret)

{

char \*err = strstr(ret,"??? ");

if(err)

{

strcpy(ret,err);

return(1);

}

}

return(0);

}

else

{

// OCTAVE mode:

return(1);

}

}

// --- Get string variable from engine ---

int mlink\_get\_var\_str(TMLink \*lnk,char \*var\_name,char \*\*data)

{

// no data yet

\*data = NULL;

if(lnk->mode == ML\_MATLAB)

{

// MATLAB mode:

// try to obtain variable

mxArray \*arr = engGetVariable(lnk->eng,var\_name);

if(!arr)

{

// not exist!

return(1);

}

if(!mxIsChar(arr))

{

// not char array!

mxDestroyArray(arr);

return(1);

}

// get var dimensions

size\_t cols = mxGetN(arr);

size\_t rows = mxGetM(arr);

// allocate string buffer

\*data = (char\*)malloc((rows\*cols + 1)\*sizeof(char));

\*\*data = '\0';

// try to read string data

mxGetString(arr,\*data,cols\*rows + 1);

// get rid of the array

mxDestroyArray(arr);

return(0);

}

else

{

// OCTAVE mode:

return(1);

}

}

// --- Get real vector variable from engine ---

int mlink\_get\_var\_dbl\_vec(TMLink \*lnk,char \*var\_name,double \*\*data,int \*size)

{

// no data yet

\*data = NULL;

if(lnk->mode == ML\_MATLAB)

{

// MATLAB mode:

// try to obtain variable

mxArray \*arr = engGetVariable(lnk->eng,var\_name);

if(!arr)

{

// not exist!

return(1);

}

if(!mxIsNumeric(arr) || mxIsComplex(arr))

{

// not numeric array!

mxDestroyArray(arr);

return(1);

}

// get dimensions

size\_t dimn = mxGetNumberOfDimensions(arr);

mwSize \*sz = mxGetDimensions(arr);

// total elements

size\_t num = mxGetNumberOfElements(arr);

if(dimn>2 || (dimn == 2 && sz[0]>1 && sz[1]>1))

{

// not 1D

mxDestroyArray(arr);

return(1);

}

// array data pointer

void \*dptr = mxGetData(arr);

// allocate string buffer

\*data = (double\*)malloc(num\*sizeof(double));

if(mxIsDouble(arr))

{

// --- double array

// copy array

memcpy((void\*)\*data,dptr,num\*sizeof(double));

// total elements

\*size = (int)num;

}

else if(mxIsInt32(arr))

{

// --- int32 array - cast to double and no bitching...

int \*iptr = (int\*)dptr;

for(int k = 0;k < num;k++)

(\*data)[k] = (double)\*iptr++;

// total elements

\*size = (int)num;

}

else

{

// --- not supported

free((void\*)\*data);

\*data = NULL;

\*size = 0;

mxDestroyArray(arr);

return(1);

}

// get rid of the array

mxDestroyArray(arr);

return(0);

}

else

{

// OCTAVE mode:

return(1);

}

}

// --- Get int32 vector variable from engine ---

int mlink\_get\_var\_int\_vec(TMLink \*lnk,char \*var\_name,int \*\*data,int \*size)

{

// no data yet

\*data = NULL;

if(lnk->mode == ML\_MATLAB)

{

// MATLAB mode:

// try to obtain variable

mxArray \*arr = engGetVariable(lnk->eng,var\_name);

if(!arr)

{

// not exist!

return(1);

}

if(!mxIsNumeric(arr) || mxIsComplex(arr))

{

// not numeric array!

mxDestroyArray(arr);

return(1);

}

// get dimensions

size\_t dimn = mxGetNumberOfDimensions(arr);

mwSize \*sz = mxGetDimensions(arr);

// total elements

size\_t num = mxGetNumberOfElements(arr);

if(dimn>2 || (dimn == 2 && sz[0]>1 && sz[1]>1))

{

// not 1D

mxDestroyArray(arr);

return(1);

}

// array data pointer

void \*dptr = mxGetData(arr);

// allocate string buffer

\*data = (int\*)malloc(num\*sizeof(int));

if(mxIsDouble(arr))

{

// --- double array - cast to int32

double \*ptr = (double\*)dptr;

for(int k = 0;k < num;k++)

(\*data)[k] = (int)\*ptr++;

// total elements

\*size = (int)num;

}

else if(mxIsInt32(arr))

{

// --- int32 array

// copy array

memcpy((void\*)\*data,dptr,num\*sizeof(int));

// total elements

\*size = (int)num;

}

else

{

// --- not supported

free((void\*)\*data);

\*data = NULL;

\*size = 0;

mxDestroyArray(arr);

return(1);

}

// get rid of the array

mxDestroyArray(arr);

return(0);

}

else

{

// OCTAVE mode:

return(1);

}

}

## m\_link.h header file:

//==============================================================================

//

// Title: matlab\_module.h

// Purpose: A short description of the interface.

//

// Created on: 2.3.2014 at 17:46:31 by .

// Copyright: . All Rights Reserved.

//

//==============================================================================

#ifndef \_\_mlink\_H\_\_

#define \_\_mlink\_H\_\_

#ifdef \_\_cplusplus

extern "C" {

#endif

//==============================================================================

// Include files

#include "cvidef.h"

#include "engine.h"

//==============================================================================

// Constants

#define ML\_MATLAB 0

#define ML\_OCTAVE 1

//==============================================================================

// Types

typedef struct{

int mode;

Engine \*eng;

}TMLink;

//==============================================================================

// External variables

//==============================================================================

// Global functions

int mlink\_init(TMLink \*lnk,int mode);

int mlink\_start(TMLink \*lnk);

int mlink\_close(TMLink \*lnk,int close);

int mlink\_cmd(TMLink \*lnk,char \*cmd,char \*ret,int retmax);

int mlink\_get\_var\_str(TMLink \*lnk,char \*var\_name,char \*\*data);

int mlink\_get\_var\_dbl\_vec(TMLink \*lnk,char \*var\_name,double \*\*data,int \*size);

int mlink\_get\_var\_int\_vec(TMLink \*lnk,char \*var\_name,int \*\*data,int \*size);

#ifdef \_\_cplusplus

}

#endif

#endif /\* ndef \_\_matlab\_module\_H\_\_ \*/

Twm\_matlab.c source file:

//==============================================================================

//

// Title: twm\_matlab.c

// Purpose: A short description of the implementation.

//

// Created on: 2.3.2014 at 20:14:31 by .

// Copyright: . All Rights Reserved.

//

//==============================================================================

//==============================================================================

// Include files

#include <ansi\_c.h>

#include <windows.h>

#include <utility.h>

#include "twm\_matlab.h"

#include "mlink.h"

//==============================================================================

// Constants

#define MAXERR 4096

//==============================================================================

// Types

//==============================================================================

// Static global variables

//==============================================================================

// Static functions

//==============================================================================

// Global variables

//==============================================================================

// Global functions

//------------------------------------------------------------------------------

// Initialize TWM link - set TWM functions path

//

// lnk - Matlab link session

// path - TWM functions root folder

// errstr - allocates and returns error string, if no error, returns NULL

//

// NOTE: Do not forget to free the allocated buffers!

//

int twm\_init(TMLink \*lnk,char \*\*errstr,char \*path)

{

char cmd[MAX\_PATH+100];

int err;

// allocate error string

char \*errbuf = (char\*)malloc(MAXERR\*sizeof(char));

if(errstr)

\*errstr = errbuf;

// add TWM root path

sprintf(cmd,"addpath('%s');",path);

err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// add TWM info path

sprintf(cmd,"addpath('%s\\info');",path);

err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// add TWM info path

sprintf(cmd,"addpath('%s\\qwtb');",path);

err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// no errors, loose error buffer

free((void\*)errbuf);

if(errstr)

\*errstr = NULL;

return(0);

}

//------------------------------------------------------------------------------

// TWM: load result(s) info(s)

//

// lnk - Matlab link session

// errstr - allocates and returns error string, if no error, returns NULL

// path - measurement root folder

// alg\_id - string id of QWTB algorithms ("": last calculated)

// res\_exist - returns non-zero if selection is valid

// res\_files - allocates and returns pointer to list of result files,

// csv string separated by tabs

// alg\_list - allocates and returns pointer to list of calc. algorithms

// csv string separated by tabs

// chn\_list - allocates and returns pointer to list of possible ref. channels

// csv string separated by tabs

//

// NOTE: Do not forget to free the allocated buffers!

//

int twm\_get\_result\_info(TMLink \*lnk,char \*\*errstr,char \*path,char \*alg\_id,

int \*res\_exist,char \*\*res\_files, char \*\*alg\_list, char \*\*chn\_list)

{

char cmd[MAX\_PATH+200];

// build command

sprintf(cmd,"[res\_files, res\_exist, alg\_list, chn\_list] = qwtb\_get\_results\_info('%s','%s');",path,alg\_id);

// allocate error string

char \*errbuf = (char\*)malloc(MAXERR\*sizeof(char));

if(errstr)

\*errstr = errbuf;

// exec command

int err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// read result exist flag

if(res\_exist)

{

\*res\_exist = 0;

int \*buf = NULL;

int size;

mlink\_get\_var\_int\_vec(lnk,"res\_exist",&buf,&size);

if(buf && size)

{

\*res\_exist = buf[0];

free((void\*)buf);

}

}

// read result files list

if(res\_files)

{

\*res\_files = NULL;

mlink\_get\_var\_str(lnk,"res\_files",res\_files);

}

// read calculated algorithms files list

if(alg\_list)

{

\*alg\_list = NULL;

mlink\_get\_var\_str(lnk,"alg\_list",alg\_list);

}

// read channels list

if(chn\_list)

{

\*chn\_list = NULL;

mlink\_get\_var\_str(lnk,"chn\_list",chn\_list);

}

// no errors, loose error buffer

free((void\*)errbuf);

if(errstr)

\*errstr = NULL;

return(0);

}

//------------------------------------------------------------------------------

// TWM: load result data

//

// lnk - Matlab link session

// path - measurement root folder

// res\_id - ID of the result file to select (-1: last, 0: average, >0: IDs of files)

// alg\_id - string id of QWTB algorithms ("": last calculated)

// cfg - display setup structure

// cfg.max\_dim - max shown dim (0: scalar, 1: vectors, 2: matrices)

// cfg.max\_array - max vector size to be shown

// cfg.group\_mode - grouping mode (0: quantities, 1: channels)

// cfg.unc\_mode - uncertainty display mode (0: none, 1: val±unc, 2: val;unc)

// cfg.phi\_mode - phase display mode (0: ±pi [rad], 1: 0-2pi [rad], 2: ±180 [deg], 3: 0-360 [deg])

// cfg.phi\_ref\_chn - reference channel is (0: none, >0: channel ids)

// csv - allocates and returns pointer to list 2D CSV table of results,

// tabs separator

//

// NOTE: Do not forget to free the allocated buffers!

//

int twm\_get\_result\_data(TMLink \*lnk,char \*\*errstr,char \*path,int res\_id,char \*alg\_id,TResCfg \*cfg,

char \*\*csv)

{

char cmd[MAX\_PATH+1024];

// build command

sprintf(cmd,"cfg = struct();\n"

"cfg.max\_dim = %d;\n"

"cfg.max\_array = %d;\n"

"cfg.group\_mode = %d;\n"

"cfg.unc\_mode = %d;\n"

"cfg.phi\_mode = %d;\n"

"cfg.phi\_ref\_chn = %d;\n"

"[csv, desc, var\_names, chn\_index] = qwtb\_get\_results('%s', %d, '%s', cfg);",

cfg->max\_dim,cfg->max\_array,cfg->group\_mode,cfg->unc\_mode,cfg->phi\_mode,cfg->phi\_ref\_chn,

path,res\_id,alg\_id);

// allocate error string

char \*errbuf = (char\*)malloc(MAXERR\*sizeof(char));

if(errstr)

\*errstr = errbuf;

// exec command

int err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// read result files list

if(csv)

{

\*csv = NULL;

mlink\_get\_var\_str(lnk,"csv",csv);

}

// no errors, loose error buffer

free((void\*)errbuf);

if(errstr)

\*errstr = NULL;

return(0);

}

//------------------------------------------------------------------------------

// TWM: call algorithms passing through qwtb.exec.algorithm.m of matlab

//

// TWM: Executes QWTB algorithm based on the setup from meas. session

// inputs:

// \*\*errstr - error string, autoallocates data if error

// \*path - path to measurement file session.info

// \*calc\_unc - override uncertainty calculation mode (default "")

// is\_last\_avg - is last record from repeated group?

// avg\_id - record id

// group\_id - group id

//

// NOTE: Do not forget to free the allocated buffers!

//

int twm\_exec\_algorithm(TMLink \*lnk,char \*\*errstr,char \*path, char \*calc\_unc, int is\_last\_avg, int avg\_id, int group\_id)

{

char cmd[MAX\_PATH+1024];

// build command

sprintf(cmd, "qwtb\_exec\_algorithm('%s', '%s', %d, %d, %d);", path, calc\_unc, is\_last\_avg, avg\_id, group\_id);

// allocate error string

char \*errbuf = (char\*)malloc(MAXERR\*sizeof(char));

if(errstr)

\*errstr = errbuf;

// exec command

int err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// no errors, loose error buffer

free((void\*)errbuf);

if(errstr)

\*errstr = NULL;

return(0);

}

//------------------------------------------------------------------------------

// TWM: get list of algorithms

//

// inputs:

// \*\*errstr - error string, autoallocates data if error

// \*\*alg\_ids - list of algorithms IDs (NULL if not needed)

// \*\*alg\_names - list of algorithm names (NULL if not needed)

//

// NOTE: Do not forget to free the allocated buffers!

//

int twm\_get\_alg\_list(TMLink \*lnk,char \*\*errstr,char \*\*alg\_ids,char \*\*alg\_names)

{

char cmd[1024];

// build command

sprintf(cmd,"[ids, names] = qwtb\_load\_algorithms('qwtb\_list.info');");

// allocate error string

char \*errbuf = (char\*)malloc(MAXERR\*sizeof(char));

if(errstr)

\*errstr = errbuf;

// exec command

int err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// load algorithm ids:

if(alg\_ids)

{

\*alg\_ids = NULL;

mlink\_get\_var\_str(lnk,"ids",alg\_ids);

}

// load algorithm names:

if(alg\_names)

{

\*alg\_ids = NULL;

mlink\_get\_var\_str(lnk,"names",alg\_names);

}

// no errors, loose error buffer

free((void\*)errbuf);

if(errstr)

\*errstr = NULL;

return(0);

}

//------------------------------------------------------------------------------

// TWM: get algorithm info

//

// inputs:

// \*alg\_id - algorithm ID string

// \*\*errstr - error string, autoallocates data if error

// \*\*par\_tab - 2D table of parameters to be displayed in Table (auto alloc, NULL if not needed)

// \*\*par\_list - list of names of parameters (auto alloc, NULL if not needed)

// \*has\_ui - algorithm has U and I inputs (auto alloc, NULL if not needed)

// \*is\_diff - algorithm suports differential mode (auto alloc, NULL if not needed)

// \*is\_multi - algorithm can process more records at once (auto alloc, NULL if not needed)

// \*unc\_guf - algorithm can estimate uncertainty (auto alloc, NULL if not needed)

// \*unc\_mcm - algorithm can calculate uncertainty by Monte Carlo (auto alloc, NULL if not needed)

//

//

// NOTE: Do not forget to free the allocated buffers!

//

int twm\_get\_alg\_info(TMLink \*lnk,char \*alg\_id,char \*\*errstr,char \*\*par\_tab,char \*\*par\_list,

int \*has\_ui,int \*is\_diff,int \*is\_multi,int \*unc\_guf,int \*unc\_mcm)

{

char cmd[1024];

// build command

//[alginfo,ptab,input\_params,is\_multi\_inp,is\_diff,has\_ui,unc\_guf,unc\_mcm] = qwtb\_load\_algorithm(alg\_id)

sprintf(cmd,"[alginfo,ptab,par,is\_multi\_inp,is\_diff,has\_ui,unc\_guf,unc\_mcm] = qwtb\_load\_algorithm('%s');\n"

"flag = int32([has\_ui,is\_diff,is\_multi\_inp,unc\_guf,unc\_mcm]);",alg\_id);

// allocate error string

char \*errbuf = (char\*)malloc(MAXERR\*sizeof(char));

if(errstr)

\*errstr = errbuf;

// exec command

int err = mlink\_cmd(lnk,cmd,errbuf,MAXERR);

if(err)

return(1);

// get flags

int \*flag;

int size;

mlink\_get\_var\_int\_vec(lnk,"flag",&flag,&size);

if(has\_ui)

\*has\_ui = flag[0];

if(is\_diff)

\*is\_diff = flag[1];

if(is\_multi)

\*is\_multi = flag[2];

if(unc\_guf)

\*unc\_guf = flag[3];

if(unc\_mcm)

\*unc\_mcm = flag[4];

free((void\*)flag);

// load parameters table:

if(par\_tab)

{

\*par\_tab = NULL;

mlink\_get\_var\_str(lnk,"ptab",par\_tab);

}

// load parameter names

if(par\_list)

{

\*par\_list = NULL;

mlink\_get\_var\_str(lnk,"par",par\_list);

}

// no errors, loose error buffer

free((void\*)errbuf);

if(errstr)

\*errstr = NULL;

return(0);

}

//------------------------------------------------------------------------------

// TWM: write record session in TWM format

//

// inputs:

// \*meas\_path - root folder of measurement session

// \*inf - session structure

//

int twm\_write\_session(char \*meas\_path, TTWMssnInf \*inf)

{

// build sessio.info path

char tsi[TWMMAXSTR];

merge\_path(tsi, meas\_path, TWMSSNINFO);

// create folder chain

create\_fld\_chain(tsi,0);

// create measurement file

FILE \*fw = fopen(tsi, "wt");

fprintf(fw,"====== COMMON SETUP ======\n\n");

info\_write\_text\_column(fw, "channel descriptors", inf->chn\_idns, inf->chn\_count);

info\_write\_text\_column(fw, "auxiliary HW descriptors", NULL, 0);

info\_write\_int(fw, "channels count", inf->chn\_count);

info\_write\_string(fw, "sample data format", "mat-v4");

info\_write\_string(fw, "sample data variable name", "y");

info\_write\_int(fw, "groups count", 1);

info\_write\_int(fw, "temperature available", 0);

info\_write\_int(fw, "temperature log available", 0);

fprintf(fw,"\n====== SETUP(S) FOR AVERAGE GROUPS ======\n\n");

fprintf(fw,"#startsection:: measurement group 1\n\n");

info\_write\_int(fw, "repetitions count", 1);

info\_write\_int(fw, "samples count", inf->N);

info\_write\_int(fw, "bit resolution", 24); // to take from setup

info\_write\_dbl(fw, "sampling rate [Sa/s]", inf->fs);

info\_write\_dbl\_row(fw, "voltage ranges [V]", inf->chn\_rng, inf->chn\_count);

if(inf->aperture)

info\_write\_dbl\_row(fw, "aperture [s]", &inf->aperture, 1);

// build record path

char rec\_name[TWMMAXTR][TWMMAXSTR];

strcpy((char\*)rec\_name, "RAW\\G0001-A0001.mat");

info\_write\_text\_column(fw, "record sample data files", rec\_name, 1);

// MAT file full path

char rec\_pth[TWMMAXSTR];

merge\_path(rec\_pth, meas\_path, (char\*)rec\_name);

info\_write\_int\_column(fw, "record samples counts", &inf->N, 1);

double Ts = 1/inf->fs;

info\_write\_dbl\_column(fw, "record time increments [s]", &Ts, 1);

info\_write\_dbl\_row(fw, "record sample data gains [V]", inf->chn\_gains, inf->chn\_count);

info\_write\_dbl\_row(fw, "record sample data offsets [V]", inf->chn\_offs, inf->chn\_count);

info\_write\_dbl\_row(fw, "record relative timestamps [s]", inf->time\_stamps, inf->chn\_count);

fprintf(fw,"\n#endsection:: measurement group 1\n\n");

fprintf(fw, "====== MEASUREMENT SETUP CONFIGURATION ======\n\n");

fprintf(fw, "#startsection:: measurement setup configuration\n\n");

info\_write\_string(fw, "digitizer corrections path", inf->dig\_corr);

info\_write\_text\_column(fw, "transducer paths", inf->tr\_corr, inf->tr\_count);

info\_write\_int\_column(fw, "channel phase indexes", inf->tr\_phase, inf->tr\_count);

char str[TWMMAXTR][TWMMAXSTR];

for(int i = 0;i < inf->tr\_count;i++)

{

if(inf->tr\_map[i][1])

sprintf(str[i],"%d;%d",inf->tr\_map[i][0],inf->tr\_map[i][1]); /\* differential channel \*/

else

sprintf(str[i],"%d",inf->tr\_map[i][0]); /\* single-ended channel \*/

}

info\_write\_text\_column(fw, "transducer to digitizer channels mapping", str, inf->tr\_count);

fprintf(fw, "\n#endsection:: measurement setup configuration\n\n");

fclose(fw);

// write MAT file with sample data

twm\_write\_mat(rec\_pth,"y",inf->chn\_data\_type,inf->chn\_count,inf->N,(void\*\*)inf->chn\_data);

// write processing info

twm\_write\_proc\_info(meas\_path, &inf->qwtb);

return(0);

}

//------------------------------------------------------------------------------

// TWM: write processing info

// Note: NOT CALLED DIRECTLY - CALLED FROM twm\_write\_session()

//

// inputs:

// \*meas\_path - root folder of measurement session

// \*qwtb - QWTB processing setup structure

//

int twm\_write\_proc\_info(char \*meas\_path, TTWMqwtbCfg \*qwtb)

{

// build sessio.info path

char qwin[MAX\_PATH];

merge\_path(qwin, meas\_path, TWMQWTBINFO);

// create folder chain

create\_fld\_chain(qwin,0);

// create qwtb.info file

FILE \*fw = fopen(qwin, "wt");

fprintf(fw,"====== QWTB processing setup ======\n\n");

fprintf(fw,"#startsection:: QWTB processing setup\n\n");

info\_write\_string(fw, "algorithm id", qwtb->alg\_id);

info\_write\_int(fw, "calculate whole average at once", qwtb->all\_rec);

info\_write\_string(fw, "uncertainty mode", qwtb->unc\_mode);

info\_write\_dbl(fw, "coverage interval [%]", qwtb->loc);

info\_write\_int(fw, "monte carlo cycles", qwtb->mcm\_cyc);

info\_write\_text\_column(fw, "list of parameter names", qwtb->par\_names, qwtb->par\_count);

for(int k = 0;k < qwtb->par\_count;k++)

info\_write\_text\_column(fw, qwtb->par\_names[k], &qwtb->par\_data[k], 1);

fprintf(fw,"\n#endsection:: QWTB processing setup\n");

fclose(fw);

return(0);

}

//------------------------------------------------------------------------------

// TWM: write sample data to MAT file

//

// inputs:

// \*path - MAT file path

// \*name - variable name to be stored

// fmt - data element format code (TWMMATFMT\_???: SGL, DBL, I32, I16)

// chn\_count - number of channels to be stored

// smpl\_count - number of samples to be stored

// \*\*data - 2D array of samples data (array of pointers to arrays belonging each channel)

//

int twm\_write\_mat(char \*path,char \*name,int fmt,int chn\_count,int smpl\_count,void \*\*data)

{

// create folder chain

create\_fld\_chain(path,0);

// create binary file for writting

FILE \*fw = fopen(path,"wb");

// store data type flag

fwrite((void\*)&fmt,4,1,fw);

// write channels count (rows)

fwrite((void\*)&chn\_count,4,1,fw);

// write samples count (columns)

fwrite((void\*)&smpl\_count,4,1,fw);

// imaginary flag (always 0)

int imag = 0;

fwrite((void\*)&imag,4,1,fw);

// write variable name length (including '\0' terminator)

int len = (int)strlen(name) + 1;

fwrite((void\*)&len,4,1,fw);

// write the variable name + '\0' terminator

fwrite((void\*)name,len,1,fw);

// now we can write data, per rows, so the sample data are written horizontally!

switch(fmt)

{

case TWMMATFMT\_DBL:

double \*\*dbl = (double\*\*)data;

for(int s = 0;s < smpl\_count;s++)

for(int c = 0;c < chn\_count;c++)

fwrite((void\*)&dbl[c][s],sizeof(double),1,fw);

break;

case TWMMATFMT\_SGL:

float \*\*sgl = (float\*\*)data;

for(int s = 0;s < smpl\_count;s++)

for(int c = 0;c < chn\_count;c++)

fwrite((void\*)&sgl[c][s],sizeof(float),1,fw);

break;

case TWMMATFMT\_I32:

int \*\*i32 = (int\*\*)data;

for(int s = 0;s < smpl\_count;s++)

for(int c = 0;c < chn\_count;c++)

fwrite((void\*)&i32[c][s],4,1,fw);

break;

case TWMMATFMT\_I16:

short \*\*i16 = (short\*\*)data;

for(int s = 0;s < smpl\_count;s++)

for(int c = 0;c < chn\_count;c++)

fwrite((void\*)&i16[c][s],2,1,fw);

break;

default:

return(1);

}

// close file

fclose(fw);

return(0);

}

## Sample sessions.info file:

====== COMMON SETUP ======

#startmatrix:: channel descriptors

HP3458A, channel 1

HP3458A, channel 2

#endmatrix:: channel descriptors

#startmatrix:: auxiliary HW descriptors

#endmatrix:: auxiliary HW descriptors

channels count:: 2

sample data format:: mat-v4

sample data variable name:: y

groups count:: 1

temperature available:: 0

temperature log available:: 0

====== SETUP(S) FOR AVERAGE GROUPS ======

#startsection:: measurement group 1

repetitions count:: 5

samples count:: 100000

bit resolution:: 28

sampling rate [Sa/s]:: 100000.000000000

#startmatrix:: voltage ranges [V]

1.00; 1.00

#endmatrix:: voltage ranges [V]

trigger mode:: Immediate

#startmatrix:: aperture [s]

1.4000E-6

1.4000E-6

1.4000E-6

1.4000E-6

1.4000E-6

#endmatrix:: aperture [s]

#startmatrix:: sampling mode

DCV

#endmatrix:: sampling mode

#startmatrix:: synchronization mode

MASTER-SLAVE, MASTER clocked by TIMER

#endmatrix:: synchronization mode

#startmatrix:: record sample data files

RAW\G0001-A0001.mat

RAW\G0001-A0002.mat

RAW\G0001-A0003.mat

RAW\G0001-A0004.mat

RAW\G0001-A0005.mat

#endmatrix:: record sample data files

#startmatrix:: record samples counts

100000

100000

100000

100000

100000

#endmatrix:: record samples counts

#startmatrix:: record time increments [s]

1.00000000000000E-5

1.00000000000000E-5

1.00000000000000E-5

1.00000000000000E-5

1.00000000000000E-5

#endmatrix:: record time increments [s]

#startmatrix:: record sample data gains [V]

4.2404940E-5; 4.2844103E-5

4.2404940E-5; 4.2844103E-5

4.2404940E-5; 4.2844103E-5

4.2404940E-5; 4.2844103E-5

4.2404940E-5; 4.2844103E-5

#endmatrix:: record sample data gains [V]

#startmatrix:: record sample data offsets [V]

0.0000000; 0.0000000

0.0000000; 0.0000000

0.0000000; 0.0000000

0.0000000; 0.0000000

0.0000000; 0.0000000

#endmatrix:: record sample data offsets [V]

#startmatrix:: record relative timestamps [s]

0.00000000000000000; 0.00000000000000000

0.00000000000000000; 0.00000000000000000

0.00000000000000000; 0.00000000000000000

0.00000000000000000; 0.00000000000000000

0.00000000000000000; 0.00000000000000000

#endmatrix:: record relative timestamps [s]

#startmatrix:: record absolute timestamps

2018-05-02T17:34:12.13001632690429687497

2018-05-02T16:34:13.66410398483276367185

2018-05-02T15:34:15.16519021987915039060

2018-05-02T14:34:16.67727661132812499997

2018-05-02T13:34:18.18936300277709960935

#endmatrix:: record absolute timestamps

#endsection:: measurement group 1

====== MEASUREMENT SETUP CONFIGURATION ======

#startsection:: measurement setup configuration

// Path to the digitizer correction file

digitizer corrections path:: DIGITIZER\HP3458A\_2x.info

// Paths to the transducer correction files, one row per channel

#startmatrix:: transducer paths

TRANSDUCERS\T01\dummy.info

TRANSDUCERS\T02\dummy.info

#endmatrix:: transducer paths

// Phase index to which each channel/transducer belongs (1, 2, 3, ...), one row per channel

#startmatrix:: channel phase indexes

1

1

#endmatrix:: channel phase indexes

// Mapping of the digitizer channels to the transducers:

// one row per transducer, each row contain index(es) of the attached channels (1 or 1;2, etc.)

// for single-ended connection: one index per row

// for differential connection: two indexes, first high-side, then low-side

#startmatrix:: transducer to digitizer channels mapping

1

2

#endmatrix:: transducer to digitizer channels mapping

#endsection:: measurement setup configuration