# Part I. Hardware and constants

To start, let’s look at what kind of devices we are controlling. A namespace is defined in “DEVICE.h”:

namespaceDEVICEHEADER

{

//Main folder contains all subprograms

constexternstringMMCfolder("C:/MMC\_SYSINT/");

//Include header files for all devices and tasks

#include"WinSock/MMCS\_MT.h"

#include"AudioTrans/AudioPulse\_MT.h"

#include"DDS/DDS\_MT.h"

#include"HV DIVIDER/HVDivider\_MT.h"

#include"MonitorsAI/MonitorsAI.h"

#include"PDA/PDA.h"

#include"CurrentSource/CurrentSource\_MT.h"

#include"Pause Timing/TimingPause\_MT.h"

}

Which includes all device headers and functions, and the namespace will be used at multiple places whenever it is needed. To understand how each device is working, the reader is directed to relevant subfolders for details, modification of control programs for each device is not a subject of this documentation.

Users are most often guided to a namespace “DATACONST.h” for storage and modification of default device settings, and in particular global constants used in data analysis.

namespaceDATACONST

{

//universal math const

constdoubleTPI=6.28318;

constdoubleZEEMAN=0.35;// 0.35MHz/G

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* PDA DAQ and plot settings:

/\* 7-pulse detection, set PDANshot=8 and BKGST=7;

/\* single pulse detection, set PDANshot=2 and BKGST=1;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

constunsignedshortPDANpixel=50,PDANshot=8,PDANofst=25;//PDA data size, 50 point each shot

constunsignedshortBKGST=7;//background shot id (YAG)

constunsignedshortAMNumChls=16,AMNumShots=5;//Analog monitors channels and shots

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* Magnetometry and EDM SCAN pixel grouping information

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

constunsignedshortSWPSTPMAX=20;//max number of sweep steps

constunsignedshortVertSubSec=3;//number of subsections along the atom array

constintmF[7]={3,2,1,0,-1,-2,-3};//Zeeman sublevel detection order

constunsignedintEDMinfocnt=3;//num. of EDM info counts: set 2 for m=0 population, asymmetry and other info

constunsignedintEDMGrpAvg=3;//num. of total groups for each lattice side

constunsignedintEDMGrpIdxRange[2]={3,21};//Index truncation range for group average, choose from 0~24

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* Automated EDM measurements constant settings

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

constunsignedintDialogWaitmseconds=5000;//MFC dialog auto pop up and wait time in milliseconds

constunsignedintMAG\_SCAN\_POINTS=10;//Number of steps during a single magnetometry scan

}

# Part II. Data Package

Below is the definition of a standard data package as of Oct 2012. This is a fundamental data unit for all data acquisitions and post analysis used throughout the program.

classDataPack

{

public:

//Data memebers

unsignedintTiming\_seqid;//Timing

CStringTiming\_loops;

floatCoil\_Cur[8]; //Coil currents

intHV\_Por;//High voltage

floatHV\_Vol;

floatHV\_Leak;

floatAna\_Mon[AMNumShots][AMNumChls];//Analog monitors

floatPDA\_Signal[PDANpixel][PDANshot]; //PDA

floatRoom\_Temp; //Other

//Methods

voidinit();//initiation of data pack

voidgetPanel(CMASTERDlg\*MasterDlg);//data from MASTER panel

boolWriteTXT(stringfilename);//Write DataPack from a TXT

boolReadTXT(stringfilename);//Read DataPack from a TXT file

boolWriteXLS(stringfilename);

voidAddPac(DataPack\*SecondPack);//Add data from a 2nd pack

voidDivPacByN(intNumOfPacs);//divide Pack data by N

};

# Part III. Spin Package

This package deals with average spins for multiple locations during a B field bias or gradient scan. Methods for field set point are also included. The parent class is used in magneometry (bias or gradient) dialogs and the child class is used in Master dialog for real time and post analysis. See definition below:

classSpinPack

{

public:

//common definitions

intACTSTEP;//actual number of data points during a sweep

doubleSpin[SWPSTPMAX][VertSubSec\*2+1];//array to store average spin for a sweep

//Data memebers and methods for bias scans

doubleLinearCoef[3][VertSubSec\*2];//linear fitting coefficients and Zero Crossings

doubledSdB[SWPSTPMAX-1][VertSubSec\*2+1];//derivative of Spin[][]

doubleQuadCoef[4][VertSubSec\*2];//quad fitting coefficients and extrema location

doubleSineCoef[4][VertSubSec\*2];//constrained sine fit vs precession time (ms) and B field

virtualboolReadSpinPackTXT(stringfilename)final;//read all data members from a TXT file for bias scans

//Data memebers and methods for gradient scans

doubleGradient[SWPSTPMAX][3];//array to store 2 gradients during a gradient scan

doubleGradLinCoef[3][2];//linear fitting coefficients for gradients zero crossing

virtualboolReadGradPackTXT(stringfilename)final;//read all data members from a TXT file for Grad scans

virtualvoidinit();//init all data memebers

};

classSpinPackDev:publicSpinPack

{

public:

//extended methods for bias scans

boolReadSpinTXT(stringfilename);//read average Spin from a TXT file

boolWriteSpinTXT(stringfilename);//write average Spin to a TXT file

voidFitLinear();//linear fitting of Spin[][] and set LinearCoef[][]

voidCalculateDSDB();//calculate first order derivative dS/dB

voidFitQuad();//quad fitting of dSdB[][] and set QuadCoef[][]

voidFitSine();//constrained sine fit for precession S(t[ms]) and set SineCoef[][]

boolWriteSpinPackTXT(stringfilename);//write all data members to a TXT file

//extended methods for gradient scans

voidCalculateGrad();//calculate gradients for a grad scan

voidGradFit();//linear fitting of Gradient[][] and set GradLinCoef[][]

boolWriteGradPackTXT(stringfilename);//write all data members to a TXT file for a grad scan

};

# Part IV. EDM Signal Package

This package deals with EDM signals at multiple locations during an EDM SCAN. See definition below:

classEDMSignalPack:publicDataPack

{

public:

doubleEDMSignal[PDANpixel][EDMinfocnt];//EDM signal and asymmetry info for each location

doubleEDMSignalAvg[2\*EDMGrpAvg][EDMinfocnt];//EDM signal and asymmetry average info for each group

EDMSignalPack();//constructor init

boolAnalyzeEDM();//Analyze PDA signal to get EDMSignal[][]

boolGrpAvgEDM(unsignedintIdxRange[]);//Group average of EDMSignal[][]

boolWriteTXT(stringfilename);//Write DataPack from a TXT file

boolReadTXT(stringfilename);//Read DataPack from a TXT file

};

classEDMScanPack

{

public:

std::vector<EDMSignalPack>EDMSCAN;//vector for storage of HV polarity pattern

floatParaSWP[4];//array for sweep parameters: initial, final, Step size and scan type

std::vector<SelectedChannels>Selected;//vector for plotting selected channels during a SCAN

voidinit();//reset EDMScanPack

boolGetPara(stringParaFile);//get sweep parameter from "SWPara.txt" file

boolAddShot(stringfilename);//add a single SHOT to the SCAN

//pull out selected points from EDMSCAN: type=group or pixel, index=which group or pixel?

boolGetSelectChls(unsignedinttype,unsignedintindex,unsignedintinfo);

voidWriteSelectedChls(stringfilename);//write selected channel data for a SCAN to file

};

# Part V. Sub dialogs

Sub dialogs are expansions of functionality of MMC main panel, callable by a button/menu click from main panel. They works identical to GUI components on the main panel, except their useage might be less frequent, or simply MMC main panel does not have enough space to hold them. These sub dialogs are what make MMC powerful and expandable.

So far 6 sub dialogs have been added:

* “ParaSweepDlg” for parameter sweep;
* “StateMapDlg” for 7 level map display and analysis;
* “MagnetometryDlg” for bias coil zero-crossing sweeping
* “MagnetometryGradDlg” for gradient coil sweeping
* “EDMSCANPlotDlg” for display data for a single EDM SCAN
* “LoopCtrlUIDlg” for automated master loop control user interface

# Part VI. Menus and Buttons

This is the core part of the MMC, as defined in the “MasterDlg.h” and implemented in “MasterDlg.cpp” , from top to bottom. The body of the program contains four major parts:

- Graphic user interface

- Atomic tasks

- Composite tasks

- Post analysis of data packs

The developer is suggested to look further into detailed implementation of each function, the place where most likely improvements can be made. Each data member has its source correspondence on the main panel, and each function has its relevant menu item or button click.

As for naming convention, each menu item serves as a prefix of each function.

“TaskInterface.h” is a supplementary implementation of “MasterDlg” functions, whenever the task is special and can be pull out as a separate block, to make “MasterDlg” looks simpler and more readable.

A complete list of menus and functions that have been implemented and tested to date: July 2013 is shown in the following table.

|  |  |  |
| --- | --- | --- |
| **Menu** | **Submenu** | **Function** |
| **Atomic** | Supertime Communication | Communication via Supertime Master |
| Microwave DDS | Calculate DDS data and output µw pulses |
| Audio Transitions | Calculate and output low frequency pulses |
| PDA Imaging | Data acquisition from the PDA amplifier |
| Current Source | Update low noise current sources for B coils |
| High Voltage | Measure high voltage and leakage current |
| Update Asyn Digital Lines | Set asynchronous lines, i.e. HV polarity |
| Analog monitors | Primary and secondary monitors input |
| Temperature | Measure science chamber temperature |
| 1 MHz CLK On | Turn on 1 MHz clock for pause timing |
| Pause Timing | Programmable delay of a Supertime sequence |
| Hardware Pattern Match | Compare patterns of two NI9205 AI channels |
| **Composite** | Parallel Tasking | Test multithreaded tasking of atomic operations |
| Magnetometry: Bias Scan | Analyze spin data for a bias scan |
| Load/Analyze Spin - Bias | Load/display analyzed spin for a bias scan |
| Magnetometry: Grad Scan | Analyze spin data for a gradient scan |
| Load/Analyze Spin - Grad | Load/display analyzed spin for a gradient scan |
| Spin Sweep Truncate | Truncate the DataPack list for a parameter scan |
| Auto Magnetometry\* | Stand-alone Auto Magnetometry Loop |
| EDM Single Scan\*  Au | A single EDM BLOCK |
| Automated EDM Block\* | Automated DAY RUN with N BLOCKs |
| **Database** | Save Panel Data | Save current single shot data to DataPack |
| Load Panel Data | Load DataPack to front panel |
| DataPack Listing | List analyzed DataPack for a parameter scan |
| DataPack Averaging | Load and average multiple DataPacks |
| Magnetometry Histogram\* | Histogram of B fields for a N BLOCK data set |
| Analyze EDM Scan\* | Load/Analyze EDM from a single BLOCK |
| Query EDM Databases ϯ | Conditional display EDM data among BLOCKs |
| **Help** | Source Code | Mastermind Visual Studio source code |
| Documentation | Documentation for Mastermind developers |
| Protocols – MFC | Microsoft Foundation Classes for Visual C++ |
| Protocols – Excel Link | Libraries for linking Excel to C++ |
| Protocols – NI DAQmx | PCI & USB device communication interface |
| Protocols – NI VISA | RS232 device communication interface |
| Protocols – WinSock | Windows Sockets for TCP/IP communication |
| Protocols – NTGraph | Libraries for 2D plot as ActiveX controls |
| Protocols – ALGLIB | Libraries for numerical analysis |

*All functions (unless otherwise noted) have been implemented and tested in situ in most recent atomic magnetometry experiments. Menu items marked with \* have been implemented in software and tested with simulated data. Menu item marked with* ϯ *has not been implemented.*

# Part VII. Automated EDM measurements master loop

## Section I. Key functions

**(1)** **EDM SCAN hardware control and data pre-analysis**

**Input**: scan parameters (Bz range) (steps)

**Output**: multiple EDMSignalPacks, store/plot pre-analyzed SCAN data

**Return**: emergency exit info

BOOLCMASTERDlg::AutoEDMScan(stringfileheader,double\*BzRange,unsignedintEDM\_SCAN\_STEPS)

{

\*Set the scan parameters for Bz

\*Loop with multithreaded hardware control and data acquisition

\*Data storage and pre-analysis (Ramsey-like fringe or elliptical method)

\*Display an EDM SCAN and analyzed results

}

**(2)** **Single magnetometry scan hardware control and data pre-analysis**

**Input**: scan parameters (type: t, x, y, z) (range) (steps)

**Output**: multiple DataPacks, store/plot spin data

**Return**: procession period or null field setting

BOOLCMASTERDlg::AutoMAGScan(std::stringfileheader,unsignedintMAG\_SCAN\_Type,double\*MAG\_SCAN\_Range,unsignedintMAG\_SCAN\_STEPS,double\*ScanResults)

{

\*Perform a reconfigurable magnetometry scan: hardware control and write DataPack files

\*Compute Average spin from a list of DataPacks

\*Analyze spin with sinusoidal or dSdB methods,

\*Display analyzed results, return measured B values (T or Bp, Gpy, Gpz)

}

Note this function is universal in a sense that it will detect the scan type, automatically configure all the hardware (magnetic coil current settings or pause timing) and automatically perform data analysis to extract results for a magnetometry scan (i.e. Larmor precession period from t-scan, or Bp/Gpy/Gpz from a bias magnetic field scan). Refer to the function implementation code section for more details.

**(3) Magnetometry (loop structure w/ single magnetometry scans)**

**Input**: max number of iterations, Larmor precession period threshold (in a do-while loop)

**Output**: a combination of all data from at least 1 iteration (4 single magnetometry scans)

**Return**: final field value or coil setting

BOOL CMASTERDlg::AutoMAG4(std::string fileheader, double LarmorPeriodThreshold, unsigned int MAX\_NUM\_Iterations)

{

-Use the global vector<MAG\_Setting> for data exchange within different MAG\_SCANs and with Mastermind

-Loop the magnetometry (t,x,y,z) scan until either MAX\_NUM\_Iterations or Threshold is reached: Set Bias field values and new Larmor precession period in the vector <MAG\_Setting> after each magnetometry scan; Update respective magnetic coil setting file (bias fields)

-Compute and update respective magnetic coil setting file (5 gradients) after each loop

}

**(4) Fully automated EDM measurements**

voidCMASTERDlg::OnCompositeAutomatededm()

Use functions (1) – (3) to realize the following master loop structure:;

Use LC: loop counter= maximum number of EDM SCANs performed before function (3)

Default file naming scheme in the folder “C:\MMC\_SYSINT\LAB\_DAT”

|  |  |  |
| --- | --- | --- |
| **Interpretation \ example** | **B1\_1\_t\_5.txt** | **D1\_2\_32.txt** |
| File prefix | B | D |
| Automated master loop run number | 1 | 1 |
| EDM SCAN number (LC) | NA | 2 |
| MAG Scan iteration number | 1 | NA |
| MAG Scan type | time (otherwise x,y,z) | NA |
| Shot ID | 5 | 32 |

For stand-alone “Auto Magnetometry” Loop, the file naming convention starts with “BS”, for example: **BS\_1\_t\_5.txt**

## Section II. Master loop implementation

**(1) Integrating the MMC-Supertime communication link: multi-process + multi-thread**

During an EDM or magnetometry scan, MMC-SupertimeMaster communication link is a special process that is distinct from the other hardware related threads, where the latter part is looped for multiple times. Therefore the communication link cannot be simply dumped into the hardware threads tank. A simple way to get around this is to

1. Start multiple hardware threads as usual for each SHOT;
2. Make an executable “MMCSexe.exe” for the MMC-Supertime communication link;
3. In MMC, start the executable as a SEPARATE PROCESS:

SeqPara=" "+to\_string(id)+" "+intostr3(loopnum);ShellExecute(NULL,NULL,MMCfolderC+\_T("WinSock/MMCSexe"),SeqPara.c\_str(),NULL,SW\_HIDE); //Hide popup window for the link

Note for this TCP/IP communication link, although many links can be connected simultaneously, ONLY ONE link can be processed at a time. A successive (waiting) link will be only processed only after the previous link is finished.

In contrast, multithreaded hardware control section for each SHOT looks like the following:

//Put all work needs to be done in each step

std::vector<std::thread>threads;

autopvThread=threads.begin();

threads.push\_back(std::thread(Audio,1));

threads.push\_back(std::thread(DDS\_EDM, 1, 2)); //DDS pulse sequence

threads.push\_back(std::thread(HVgetShow,this,3));//High volatge

threads.push\_back(std::thread(MonitorgetShow,this,1));//Monitors

threads.push\_back(std::thread(PDAgetShowEDM,this,k));//PDA DAQ

threads.push\_back(std::thread(TempgetShow,this));

pvThread=threads.begin();

while(pvThread!=threads.end())

{

pvThread->join();//require a C++11 or later complier platform

pvThread++;

}

Note all the above threads are running simultaneously in a single process. Subsequent codes cannot be executed until all the threads finish.

**(2) Modal vs modaless dialogs in MFC**

Modal dialogs are useful in early steps of the experiment, where tasks may vary on a daily basis and very frequent software-user interactions are needed. In the modal mode, message pump of a dialog is passed entirely to the user and background programs (MMC main panel/program) do not have access to dialog control. Example code in MMC to recall a modal dialog and waiting for user input is shown below:

EDMSCANPlotDlg\*EDMSCANDlg=newEDMSCANPlotDlg();

EDMSCANDlg->DoModal(); **//**Pop up dialog

deleteEDMSCANDlg;

In the final phase of full automation, all the previously written sub-dialogs, including dialog function and control code can be used in the modaless mode. Namely, message pump of a modaless dialog is controlled by the background MMC main program. The user can then ONLY view the dialog for real-time monitoring purpose. Example code in MMC to recall a modaless dialog and simulate a button click event is below:

//Initiate a modaless MFC window

EDMSCANPlotDlg\*EDMSCANDlg=newEDMSCANPlotDlg();

EDMSCANDlg->Create(EDMSCANPlotDlg::IDD,this);

EDMSCANDlg->ShowWindow(SW\_NORMAL);//show frame

EDMSCANDlg->UpdateWindow();//show controls

//Simulate a button click

WPARAMWParam=MAKEWPARAM(IDC\_BUTT\_EDMSCAN\_PLOTANA,BN\_CLICKED);

EDMSCANDlg->SendMessage(WM\_COMMAND,WParam,NULL);

EDMSCANDlg->UpdateWindow();//update window after click

//Pause the dialog for some time

DialogPause(DialogWaitmseconds/5);

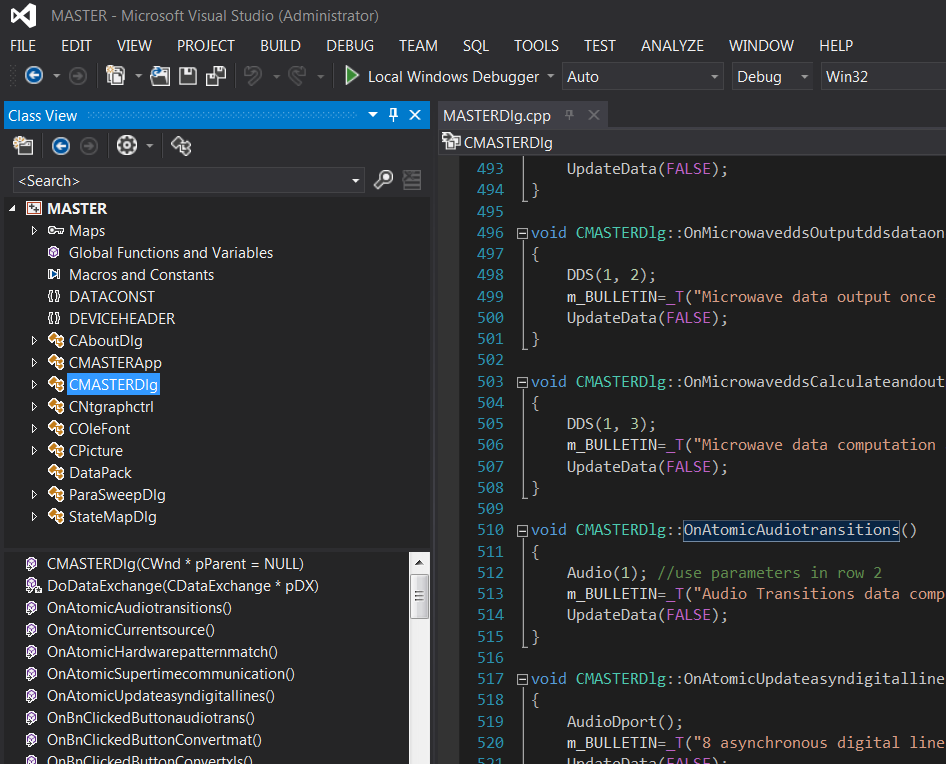
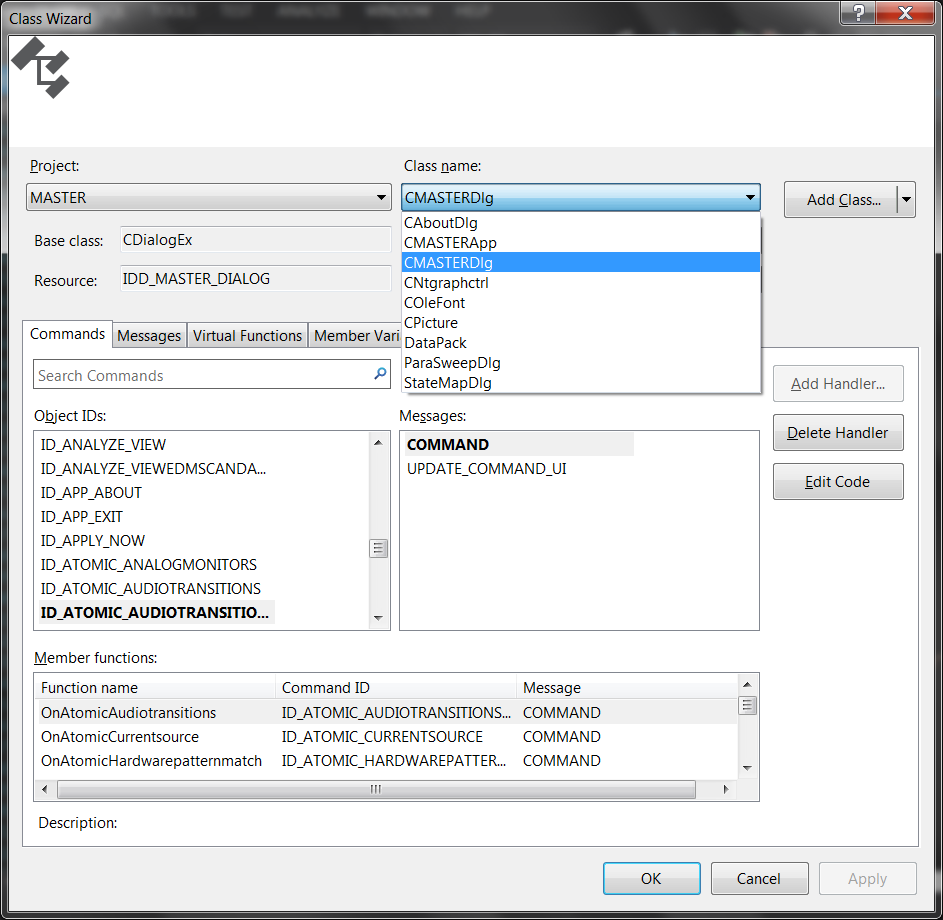
//Destroy window

EDMSCANDlg->DestroyWindow();

deleteEDMSCANDlg;

# Part VIII. Visual Studio 2012 Class Wizard

The MMC program is an internally connected structure of a list classes. As a standard tool in MSVC2012, developer can check data members and functions using “Class Wizard” or “Class View”:

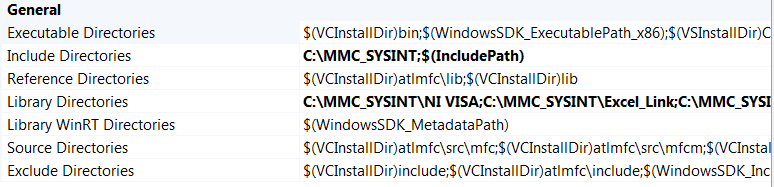


As of 2013, there are estimated 12,000 source lines of codes (hand typed, excluding source libraries) regarding code-based hardware control/DAQ, MMC panel functions/menus, dialogs, signal data structures and file I/O.

# Part IX. Environmental settings

*Developers should read this section very carefully. An incorrect setting like this could easily cause you a couple of hours of struggle.* To minimize coding effort on multithreading and GUIs, a most up-to-date Microsoft Visual Studio (2011 or later) is necessary to run the program. Visual Express does not have enough packages to fully support this program. To building a MSVC project from scratch (i.e. when there is a version upgrade or platform change):

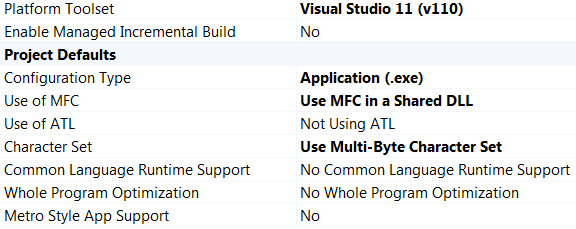
(1)”Master Properties”🡪”Configuration properties”🡪”VC++ Directories”: add MMC folder to “Include Directories”, add all MMC subfolders to “Library Directories”;



Library Directories

C:\MMC\_SYSINT\NI VISA;C:\MMC\_SYSINT\Excel\_Link;C:\MMC\_SYSINT\DAQmx;$(LibraryPath)

(2)Use “Multi-Byte Character Set” for correct font display and string manipulation.



(3)Compile with command prompt in visual studio

VS2012 has a known bug that command “cl filename.cpp” won’t work due to a missing file “mspdb110.dll”. The bug might be fixed in the future, but not yet until Jan 2013. This can be found by running “vcvarsall.bat” in the folder “C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC”, which will reset and link the correct compiler.

The command “vcvarsall.bat” may experience “ERROR: Cannot determine the location of the VS Common Tools folder”. Microsoft looks for the location of its tools by looking at the variable “VS110COMNTOOLS”. In my case this variable is set, BUT starting a VS2012 command shell fails to set this variable – because the Microsoft script unset the variable first and then query the registry for the installation path and set it again. To query the registry it uses the command “reg” and “reg” is located in c:\windows\system32. But this directory is NOT defined in the PATH variable content anymore and therefore reg is not found. The not finding of “reg” creates an error and VS110COMNTOOLS remains unset and the whole stuff is not working any more. Therefore the missing of “C:\windows\system32” in PATH is the reason for all these errors. The final solution is to add (if not exist) variables:

**VS110COMNTOOLS** = C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\Tools\

**PATH** = C:\Windows\system32;C:\Windows;

# Part X. Special note on operating platforms

As of MAY2012, I found the most up-to-date DAQmx drivers (9.5) are ONLY compatible with 32bit applications (one can locate “DAQmx.h” and “DAQmx.lib” in folder “C:\Program Files(x86)\National Instruments” on 64bit machines, but not in folder “C:\Program Files\National Instruments” ), even though all the NI software and hardware are successfully running on the Win7 64bit OS. NI is addressing this issue but a truly x64 driver has not come out yet.

<http://forums.ni.com/t5/Measurement-Studio-for-NET/Is-NI-DAQmx-supported-as-true-64-bit-mode-in-Vista-x64/td-p/689149>

This means even though most parts of code project except “NI hardware related” for this master mind program can either run 32bit or x64 mode, so far it can only work in the former.

Embedding MATLAB programs (i.e. environment settings) in this application then is limited to 32bit version of MATLAB either on a 32bit or x64 OS.

<http://stackoverflow.com/questions/9500400/unresolved-external-symbol-error-using-matlab-api>