# Part I. Mastermind introduction

Mastermind is a user-friendly C++11-LabVIEW hybrid automation software written primarily in Visual Studio to realize the hardware control and data functions as needed by the Penn State eEDM lab. As of August 2013, the estimated hand-typed source line of codes (SLOC) for Mastermind software functions is 12,000, excluding source code of commercial and open source packages standard libraries like STL, Windows Sockets and ALGLIB. The software features

(i) Adaptive hierarchy to meet variable requirements, in particular, the software is designed to work with multiple formats of atom signals and monitor data acquisitions, and universal functional interfaces enables convenient serialization and parallelization to build up complex control loops.

(ii) Multithreading routines to control hardware. Devices running in parallel (using the new C++11 “pthread” library) on the Mastermind computer fully utilize available computation power and reduce experimental dead time.

(iii) Communications with remote LabVIEW programs.

(iv) Data analysis and management with STL and ALGLIB, a C++ based numerical analysis open source with nonlinear fitting packages, matrix toolboxes, etc.

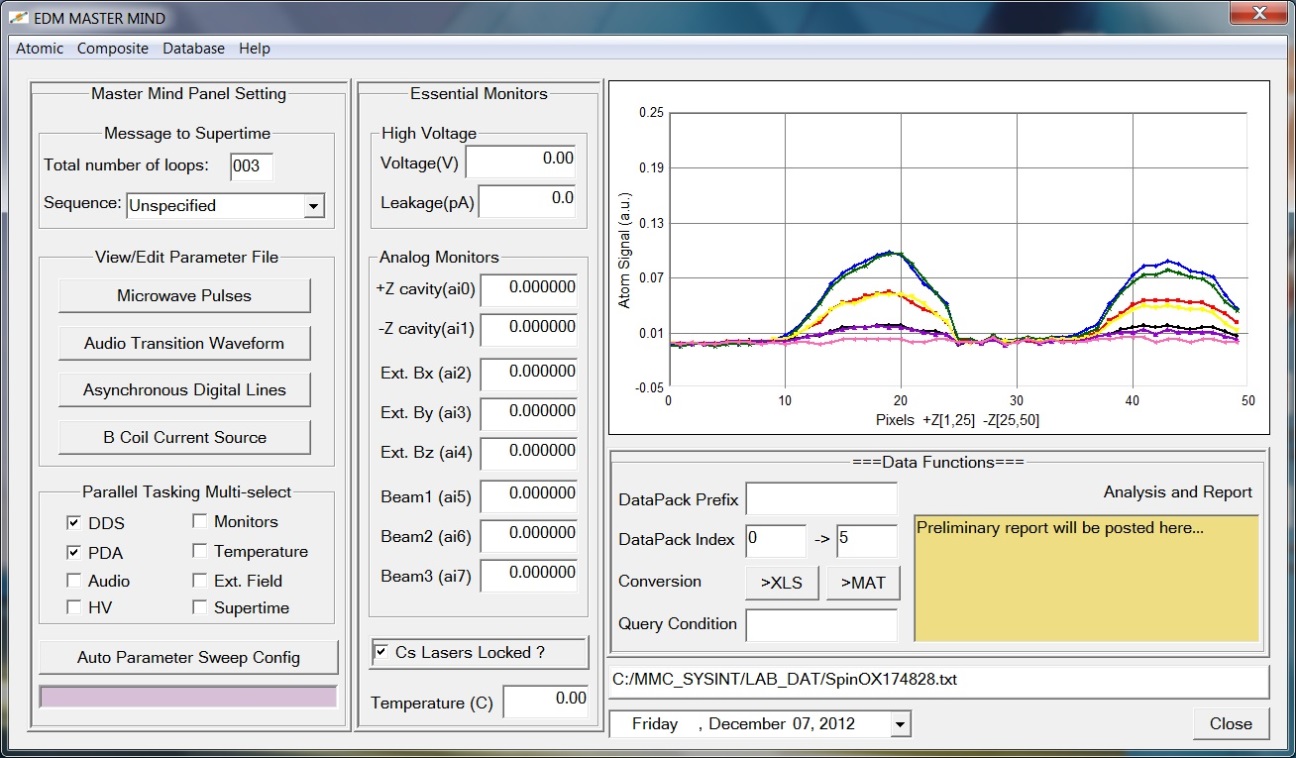


Figure. Main control panel user interface

Left: panel settings for various tasks, or links to formatted (excel) files containing settings

Middle: essential and non-essential monitors that may need to keep track of in real time

Right::above: atom signal, updated for each shot

Right::below: data functions for analysis and database tasks

# Part II. Atomic tasks

The menu items under “Atomic” (abbr. for atomic operations) allow one to trouble shoots each device or run any individual hardware related task. Below is a list of devices implemented on EDM master mind computer, each has a separate folder contain its own low level hardware access functions (“\*.h” header file) and high level operating functions (“\*\_MT.h” header file). For the ease of readability and future upgrade, each device has an independent C++ class with data members and methods, modifying the classes and high level functions allow one to run the device in any desired way. This part requires understanding how each device and protocol works, which has been completed. Documentation and links to major protocols relevant to each task is under the menu “Help”.

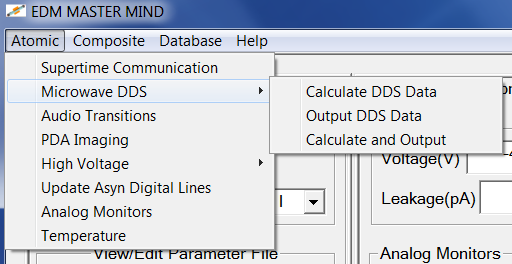


Figure. Menu for single tasks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task | Device[Interface] | Type | Protocol | Trigger | Exist? |
| Timing sequence t-scan | PCI6601[PCI] | DO | NI DAQmx | Supertime | Yes |
| DDS control | PCI6534[PCI] | DO | NI DAQmx | Supertime | Yes |
| PDA data acquisition | PCI6071E[PCI] | AI | NI DAQmx | Supertime | Yes |
| Audio transition waveform | PCI6713[PCI] | AO | NI DAQmx | Supertime | Yes |
| Low noise B field control | USB6501[USB] | DO | NI DAQmx | Software | Yes |
| HV Parity control | PCI6713DIO | DO | NI DAQmx | Software | Yes |
| HV Leakage current monitor | pA6485[RS232] | DI | NI Serial | Software | Yes |
| HV Divider monitor (opt.) | AT34401A[RS232] | DI | NI Serial | Software | Yes |
| Temperature tracker | TSci506F[USB] | DI | Excel/libxl | Software | Yes |
| Cavity power/lock monitor | NI9205[USB] | AI | NI DAQmx | Soft[S/H] | Yes |
| External B field monitor [1] |  | AI |  | Soft[S/H] | NO |
| Cs laser lock monitor [2] | PDA |  | Data Analysis | Soft[S/H] | Yes |
| Atom number monitor [3] | PDA |  | Data Analysis | Supertime | Yes |

Table. Hardware and devices on EDM master mind computer

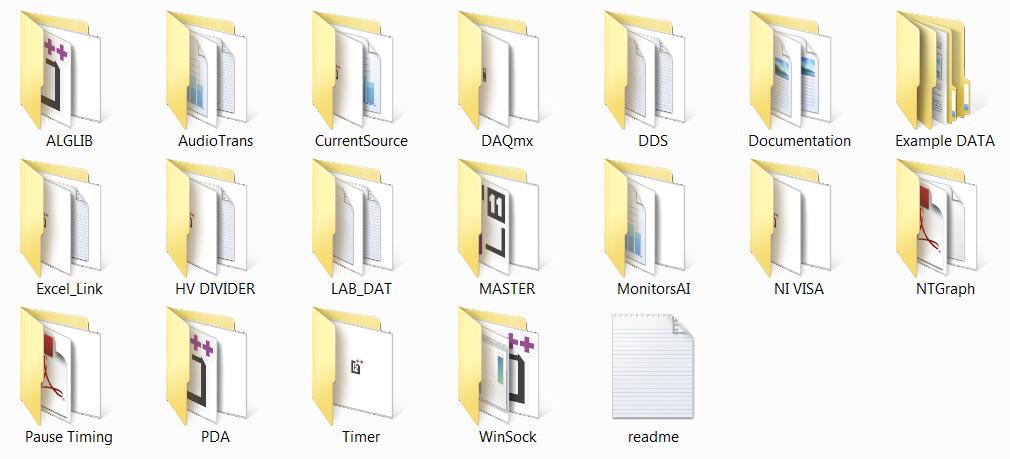
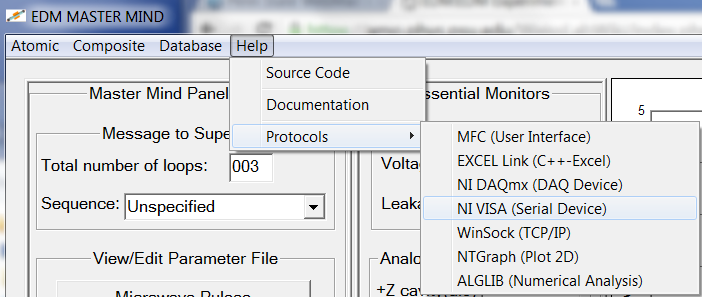


Figure. Folders for individual programs and parameter settings

# Part III. Help and documentation



Under the menu bar “Help”:

“Source Code” links to different folders containing codes for each device and master control panel. Nontrivial protocols include (all C++-style, platform independent and free) :

MFC: Microsoft foundation class libraries for graphics user interface

EXCEL\_Link: a C++ version of libxl package for excel read and write

DAQmx: National Instrument protocols for advanced DAQ device interface

VISA: National Instrument protocol for serial device interface

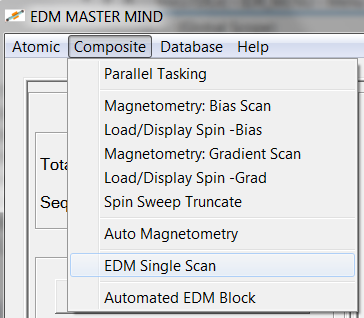
WinSock: Windows socket 2 based TCP/IP protocol

NTGraph: ActiveX control for 2D plots embedded in MFC dialogs

ALGLIB: Numerical analysis and data processing

Installation, configuration and usage of above packages (which are beyond scope for this documentation) may be found by the menu links or in their corresponding subfolder.

# Part IV. Composite tasks



Based on the part w/ “individual tasks”, a composite selection including sequential and parallel execution of different tasks following a typical experimental flow can be achieved.

## Example I: Asynchronous multi-choice parallel tasking with parameter sweep

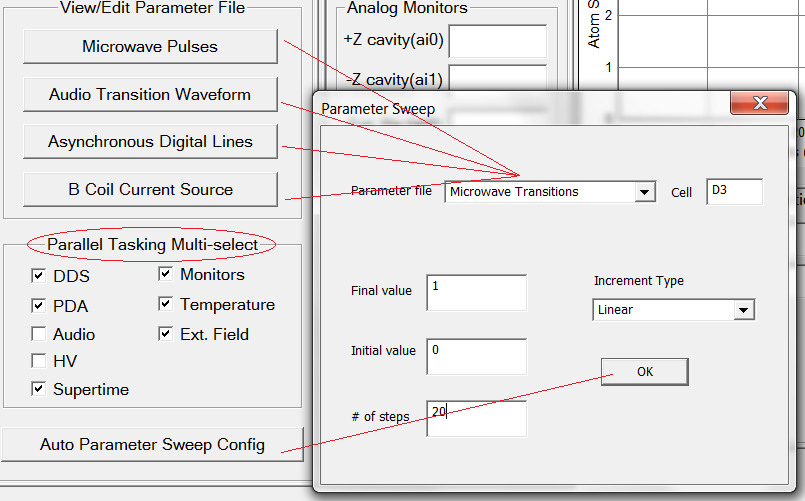


Figure: multitasking check boxes and settings for a parameter sweep. A realistic code block to execute multiple jobs in parallel looks something below. Note the function interface is either a direct copy or indirect inheritance from “Individual Tasking”.

## Example II: Data functions for local mapping

After a large set of data is taken, it is essential to look the signal vs pixel number or a group of pixels, to check EDM (or magnetic field) as a function of vertical position. This function will help mapping out local effects.

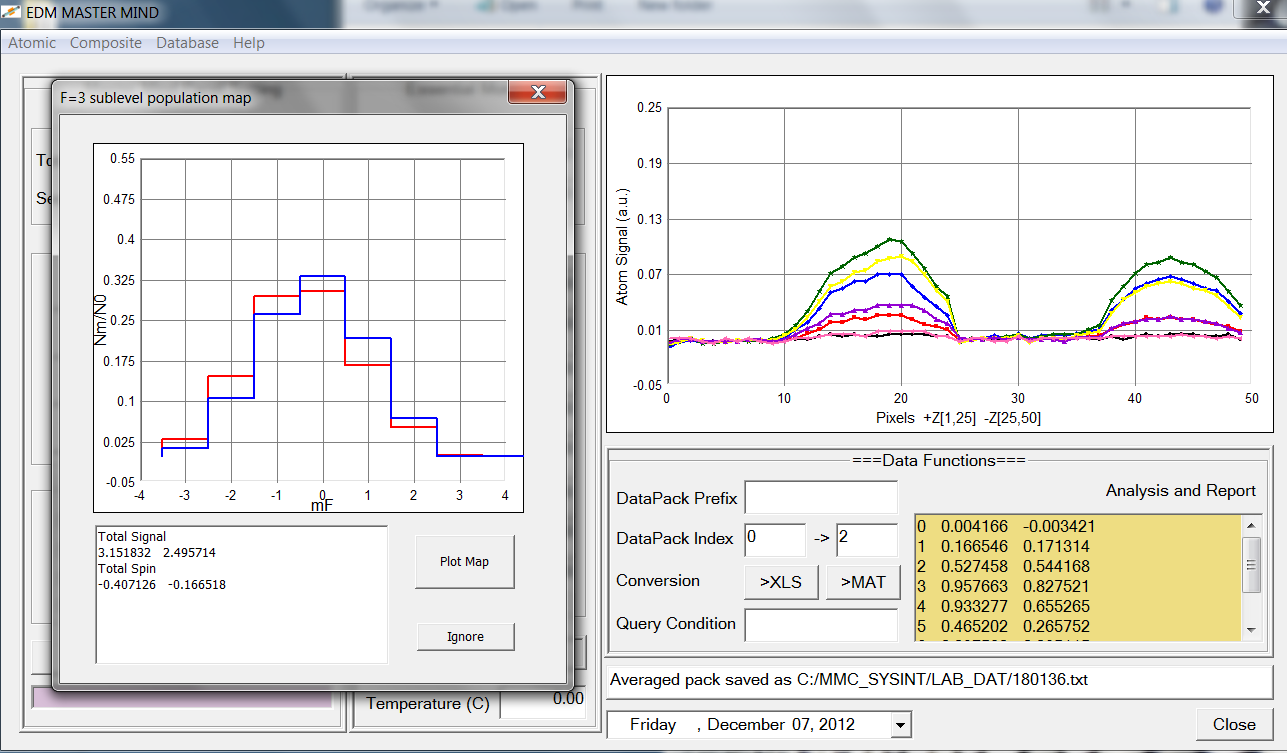
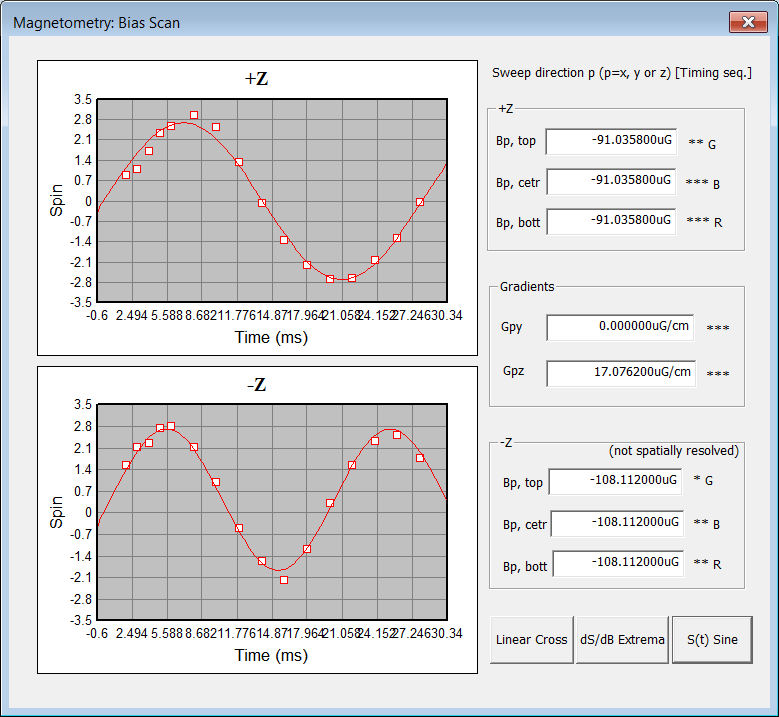
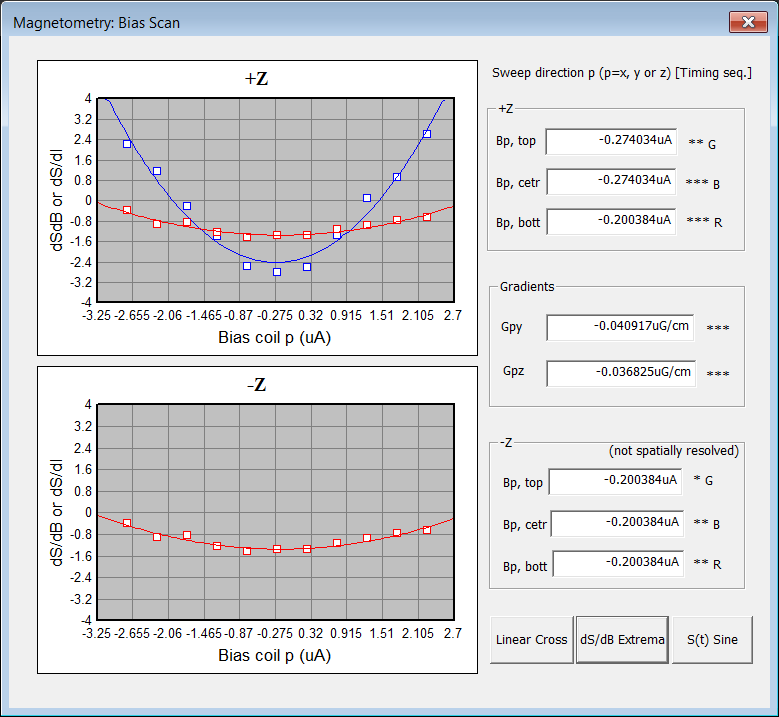
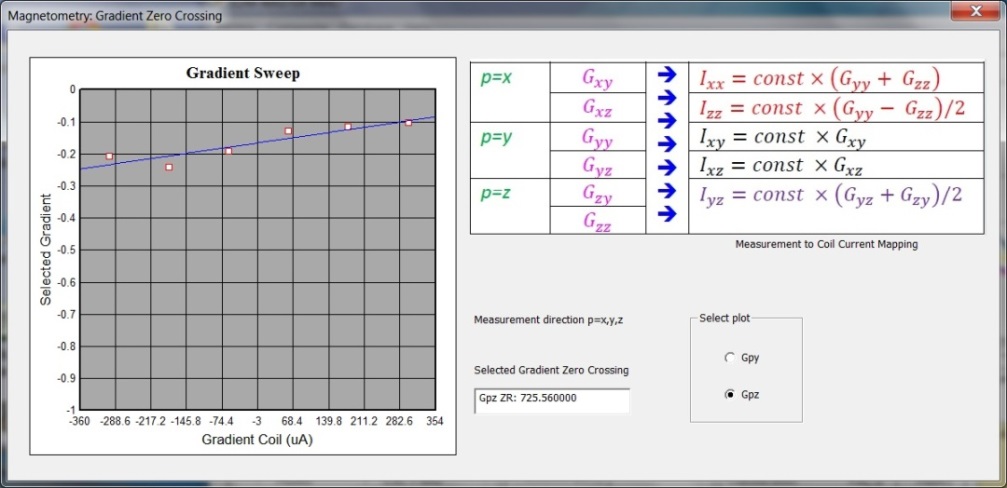


Figure: Capability of multi-state imaging and analysis.

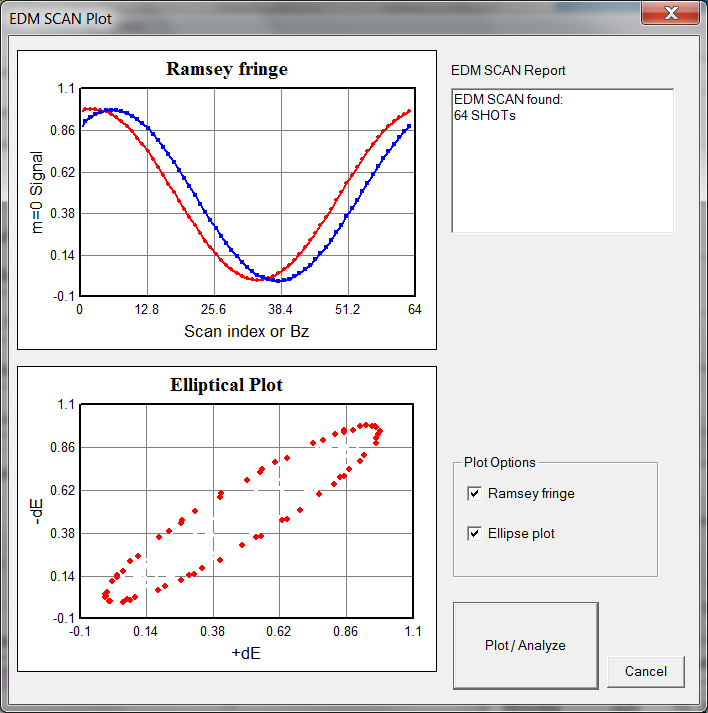
## Example III: Magnetometry bias scan: linear crossing, quad extrema and S(t) sine fit



## Example IV: Magnetometry gradient scans

****

## Example V: a single EDM SCAN



\*Click menu “EDM single scan”, pop up the “parameter sweep” dialog, set the scan parameters for Bz

\*Write a loop with multithreaded hardware control and data acquisition

\*Compute/Store an element of EDMDataPack for each shot

\*Display signal from a SCAN on chart, including both sinusoidal and elliptical plot

\*Derived class EDMDataPack from DataPack,

Added data members: EDM signal, level asymmetry

Added method: EDMDataPack .txt file I/O, calculate EDM signal and level asymmetry from PDA signal

## Example VI: Automated EDM Block

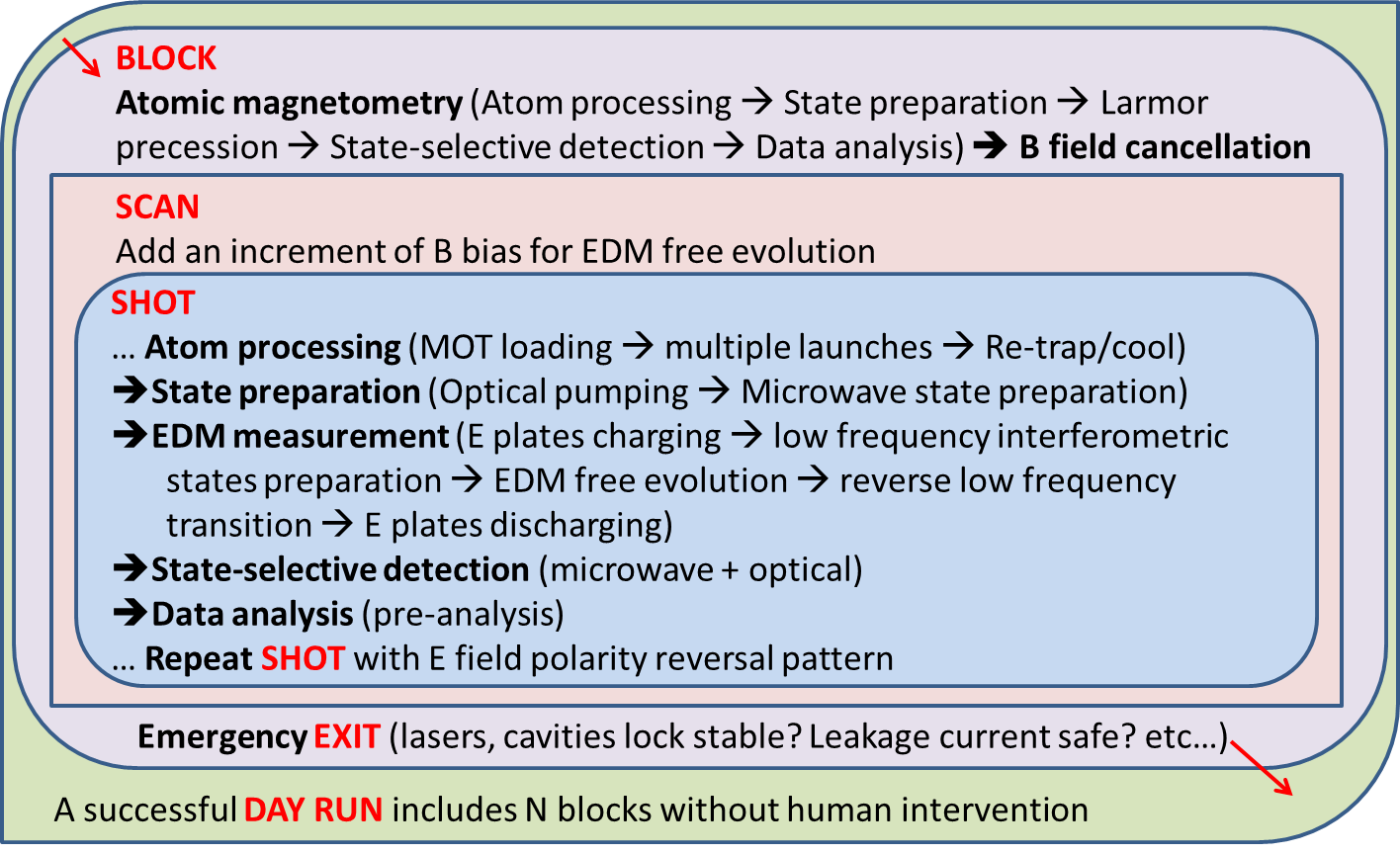


Figure. The hierarchy of automatic EDM measurements.

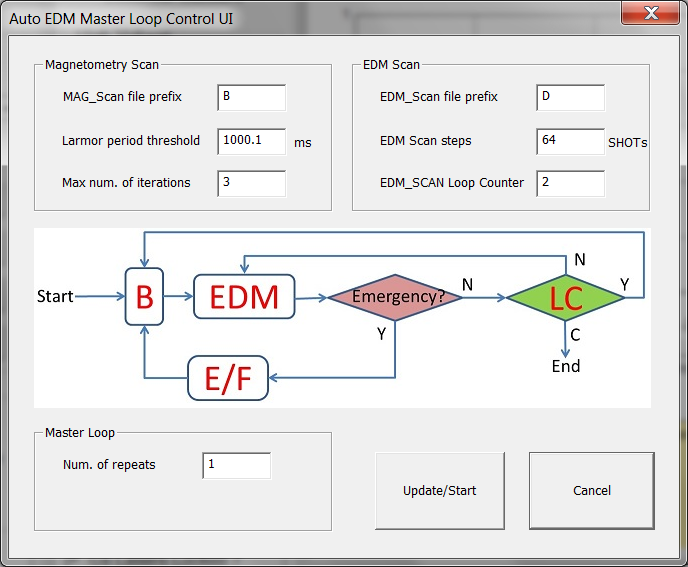
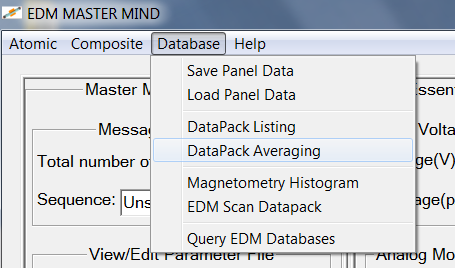


Figure. Control flow chart of automatic EDM measurements.

**B**: atomic magnetometry to measure and cancel residue magnetic fields; **S**: single shot EDM measurement during a SCAN; **LC**: loop counter; **E/F**: emergency exit and fix; **Y**(**N**): if the condition is (not) satisfied; **C**: conditional exit.

## Part V. Data management



This part deals with pulling up structured data and post analysis. Exact implementation is to be determined, most plausible and efficient method might be a C++-MATLAB struct based database: real-time partial (also necessary) analysis of the data might still be done in the framework of C++ due to heavy load of MATLAB that might slow down other hardware related tasks, but all data sets can be saved in MATLAB struct for post analysis.

Fundamental element of EDM data bin (termed EDM TWIN) may be a *C++ class* consisting a pair of:

PDA DataPack (25 pixel inverted) + Monitor Data + sequence info + EDM SignalPack (m=0 fractional population and asymmetry) + pre-analyzed EDM w/ identifying tags

With E field reversed (two successive SHOTs).

The *class* is in text format that allows fast I/O in MMC and can be viewed or edited by MMC.

Sample *C++ class* with I/O functions (i.e. DataPack) has been converted in MMC program to *MATLAB struct* and *EXCEL stand-alone file* for remote view options without MMC

Possible solutions to post processing and conditional querying of EDM data bin in the long term

-do everything with C++ in MMC through the open source library ALGLIB and SQLite

-post process and query in MATLAB with converted *MATLAB struct*

-other not-yet-identified program, i.e. a hybrid between EXCEL and MS Visual FoxPro

Speed should not be an issue for any of the above solution if the database size is below GB scale.

Methods to analyze the EDM TWIN data may vary, depending on sensitivity used and whether we can reliably park near the most sensitive point of a fringe.

Preferred solution (probably better after all) is not to use MATLAB at all in the master mind program. Instead, (1)define a purely C++ based data struct callable to MMC, and write this type of data sets to formatted files readable to MMC with fstream; (2)Write a separate C++ program that converts the text files to MATLAB struct based “.mat” files. Since most data real-time and post analysis can be done in step (1) with ALGLIB (C++ packages, platform-independent, must faster than MATLAB), step (2) may only be useful if by any chance one needs to run data analysis in MATLAB.

So far 3 types of data formats are used in the actual implementation:

1. Formatted text file ‘\*.TXT”, fast I/O, can be viewed w/ text editor;
2. Formatted Excel file, large size and modest I/O operation.
3. MATLAB struct files converted from (2), small size and modest I/O operation, only can be viewed w/ MATLAB software, and can be accessed by MATLAB programs for post analysis if necessary.

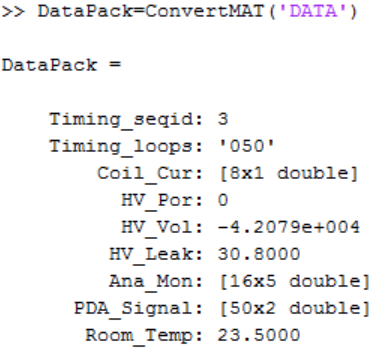


Figure: an example of DataPack members in MMC being converted to a MATLAB struct.