LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T070182-00-R *LIGO* Aug 7th, 2007

Line Monitor Web Interface Reference

Kiel Howe

Distribution of this document:
LIGO Science Collaboration

This is an internal working note
of the LIGO Project.

This material is based upon work supported by the US National Science Foundation under Cooperative Agreement No. PHY-0107417.

Acknowledgements

I worked on this project as a Virgo-LIGO 2007 summer exchange student. Thanks to my supervisor Michele Punturo, as well as to Irene Fiori and Gabriele Vajente for their guidance, feedback, and help. Thanks also to Riccardo DeSalvo and Francesco Fidecaro for all their work organizing this amazing experience.

Table of Contents

Introduction

In the attempt to reach design sensitivity, one of the primary activities of the Virgo Noise Group is identifying and eliminating persistent peaks in the Virgo dark fringe spectrum, or lines as they appear in spectrograms and other plots. Some lines have obvious explanations, like the 50hz and harmonics lines due to the power mains, but many require careful investigation and experimentation to explain. To this end, Virgo scientists have created many useful tools, including programs to calculate coherences and automatically pick out peaks, and databases to keep track of known lines and lock events. The goal of the Line Monitor Web Interface is to unify these tools in one application.

Existing Tools

Coherence is a mathematical operation in the frequency domain that establishes relationships between different signals. This summer another Virgo summer student, John Draskovic, created a C program to automatically compute coherences between the dark fringe channel and all environmental channels on a lock-by-lock basis.

LineMonitor is a C program that runs on-line, taking Fast Fourier Transforms of the dark fringe channel, and identifying peaks in each time window. The frequency and amplitude of each peak is stored, as well as the sound to noise ratio (SNR), which is the ratio of the amplitude to a computed background. Graphs of this information with respect to time can be useful to see drift in frequency and SNR, which can be related to time-dependent variables like temperature change. They can also make apparent sudden jumps that can result from events like control adjustments or earthquakes. These graphs are especially useful since Virgo Science Run One (VSR1) has started making longer periods of continuous science mode data available for analysis. Unfortunately, there was no simple way of accessing the information produced by the existing version of LineMonitor, as it wrote frames incompatible with the standard Virgo analysis software, DataDisplay. The data produced by LineMonitor is also useful for persistence calculations, which are a mathematical measure of how often a line is present in a given time range.

The known lines database is a web application using a MySQL database that allows documentation and logging of investigations of lines by their frequency, as well as cataloging of characteristics like drift. Most usefully, this database contains entries for all lines which have been cataloged so far.

The locks database is another MySQL database that contains information about the operation mode of the detector. This information was also available from a web application, but required cumbersome cutting and pasting into forms and configuration files. Easy use of this data is important because most calculations with other tools are only useful for science mode data.

Database Development

The backbone of the application is the data produced by LineMonitor. However, as mentioned earlier, the existing version did not store data in a usable form. So, it was decided to store the LineMonitor data in a MySQL database.

Storing the data in a database provides several advantages over storing in frames or another flat file. As far as frames are concerned, reading a relatively small channel over a long time range will always be much slower than a single-channel database. This is because frame data is grouped in sequential storage by its gps time, not by its channel. This translates into lots of slow seeking over irrelevant data from other channels. For a plain flat file, separate copies for different users and delays between updates would be necessary to handle concurrent reading and writing; this is something the database does very effectively behind the scenes.

Additionally, using a MySQL database allows seamless integration with the existing known line and locks databases. Powerful SQL queries can be written to join data from all three of these tables. There was also already an existing MySQL testing deployment at Virgo because of these other databases.

Modifying the LineMonitor program to write to the database was trivial. However, tuning the database's performance, with one quarter of a gigabyte of data being written for each week of VSR1 data, proved to be difficult. At one point during development, queries that were expected to take a couple of minutes were taking five times as long, making the application very unacceptably slow.

By benchmarking several different indexing methods and storage engines, it was possible to find a solution that provided good performance and scalability. By replicating the currently available VSR1 data, acceptable performance was projected out to one half of a year of data. Although the database was not tested with more than that much data, analysis of the benchmark results suggests that performance will not suffer. An in depth description of the benchmarking methodology, results and conclusions is available in Appendix D.

Interface Development

PHP running on an Apache web server was chosen as the platform for the main interface. This platform was appealing from both the user and development points of view.

For the user, there is very little learning curve, because the application behaves just like any familiar web page. For example, when the page is bookmarked, it will come up with the exact same settings and view as before, and to save a graph, one just right clicks and selects “save as.” In addition, the user need not worry about resource use, which is split between the web server and the database server.

From the development standpoint, PHP has very simple bindings to MySQL, and HTML forms are very intuitive to design. Also, because I was developing on an intranet server, I was able to regularly make the newest stable version of the application available to members of the noise group. The feedback from these stable releases was helpful in directing the development of the application at all stages.

Some drawbacks to this development platform did become evident as the project progressed. For one, because PHP can not pass large amounts of data across page loads, each request required a complete reload of data from the MySQL server, even if it was on the same data range as before. The significant performance hit of this was overcome in the interface by creating quick in-memory caching tables on the MySQL server. This is an effective solution, but makes some simple parts of the code much more complicated. PHP also has a weaker object model and poorer debugging tools when compared to many compiled languages. In retrospect, developing the GUI as a Java applet would have provided similar advantages to those of PHP, without so many disadvantages.

For a detailed reference to the web interface code, see Appendix C, which includes class diagrams and flow charts.

Component Integration

The web interface allows the user to select LineMonitor data to view by specifying a frequency range, a time range, and a minimum SNR. Any data from LineMonitor can be graphed on either the x or y axis. Graphing is done using a simple free PHP chart library. The graphing was designed to be as informative and quick as possible. This makes the interface useful for visualizing data, but not producing high quality graphs. This decision was made that rather than try to duplicate the features of the many available graphing suites, it would be better to simply provide a link to download the selected data in the portable tab separated value format, for importing into a graphing program.

Although it is called LineMonitor, the data provided by LineMonitor does not say anything about which peak belongs to which line. So, line identification in the interface is implemented in two different ways. First of all, the user can manually list frequency ranges which represent lines. Or, the interface can automatically identify lines with a minimum persistence at a desired resolution. The algorithm works by recursively dividing the frequency range into two overlapping ranges, and calculating persistences for each range. If a range has the desired persistence, it is further divided, otherwise it is thrown out. This is repeated until the desired resolution has been reached, at which point any range with the desired persistence is returned. The algorithm is effective in most cases except lines that are wider than the desired resolution, and lines that drift more than the desired resolution.

The interface calculates persistence with a simple method. It is merely the number of distinct times a peak is present in the frequency range of the line divided by the maximum possible times based on the time range being viewed and time resolution of the data.

When lines have been identified either automatically or manually, they can be matched with known lines within a specified delta frequency. Basic information about found matches is provided in a table, as well as links to the appropriate entry in the known lines database.

For coherence integration, I worked with the other summer student to adapt his program to write to another MySQL database. For each line, the interface searches the coherence database within the specified delta frequency and within the selected time range for the maximum coherence with each channel. Matches are displayed in a table with links that allow the user to view the complete coherence graph of any matched channel.

Conclusions

There were two main problems with the development process for this application. First the database benchmarking should have been the one of the very first steps in the database design. Instead, benchmarking interrupted interface development and then required changes to the interface code to deal with the new tables. Secondly, more thought should have been put into the drawbacks of using PHP for the interface. Using a more appropriate language like Java would have resulted in less complicated code and easier debugging.

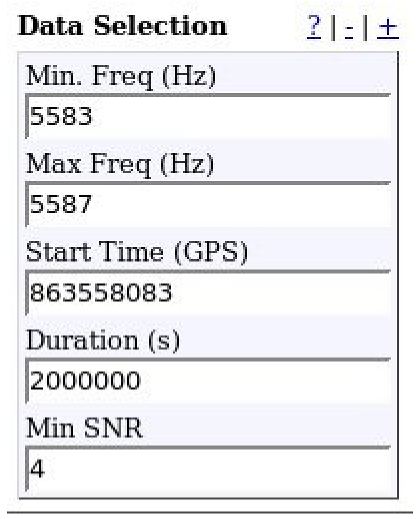
However, despite these two mistakes in development, the application has at least basic implementations of all of the desired features, performs well, and is stable. Furthermore, the application has been designed to be easily expanded, refined, and supported. The documentation contained within the appendices will be useful in all of these activities.

APPENDIX A: Web Interface Users Guide

(This users guide can also be accessed as the online help for the web interface)

Title Bar

The '+' button will expand the dialog, the '-' button will shrink the dialog to only the title, and the '?' button opens a window for the help for that dialog.



Min. Freq (Hz): 5583
Max Freq (Hz): 5587
Start Time (GPS): 863558083
Duration (s): 2000000
Min SNR: 4

Data Selection

Uses this dialog to specify the data you want to look at. The first time you look at range of data, it will be read off the disk and cached in memory, which can take one or two minutes. After that, any data you look at within that range will load quickly from the cache. If you select data outside of the range, the cache will be cleared and reloaded with new data.

Min Freq and Max Freq

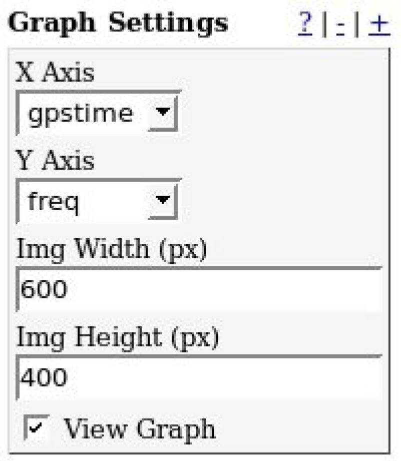
Specify the data range in hz you wish to look at. The database contains data from 1 to 1000hz with a resolution of 0.01hz.

Start Time and Duration

Start Time is the first gps time you want to see data from. Duration is how many seconds you want to see data for. You can also use the locks dialog to automatically fill in these fields.

Min SNR

Only points with SNR greater than this value will be displayed. Only points with an SNR > 4 are stored in the database. Changing the min SNR does not require reloading the memory cache from disk.



Graph Settings ? | - | +

X Axis: gpstime
Y Axis: freq
Img Width (px): 600
Img Height (px): 400
[✓] View Graph

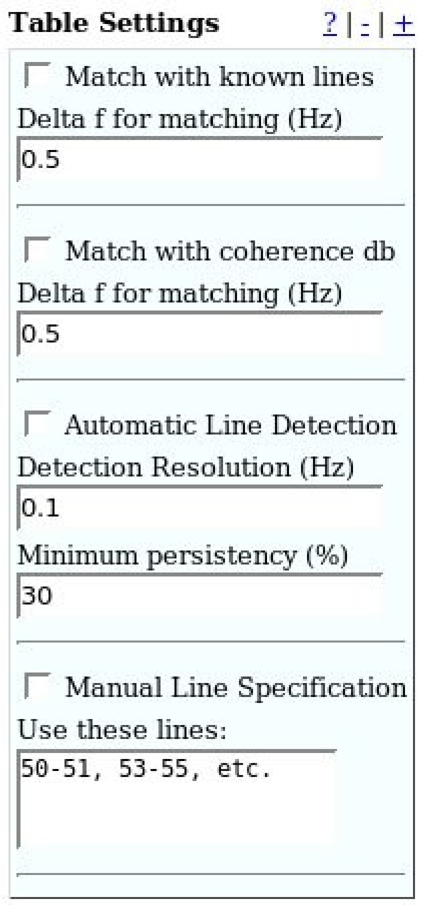


Table Settings ? | - | +

[ ] Match with known lines
Delta f for matching (Hz): 0.5

[ ] Match with coherence db
Delta f for matching (Hz): 0.5

[ ] Automatic Line Detection
Detection Resolution (Hz): 0.1
Minimum persistency (%): 30

[ ] Manual Line Specification
Use these lines: 50-51, 53-55, etc.

Graph Settings

This dialog controls how the graph of the data selected with the data selection dialog is displayed. The result is the graph window.

X Axis and Y Axis

These options control which field will be displayed on which axis. Any combination is possible.

Img Width and Img Height

These control the actual size in pixels of the image generated.

View Graph

If you do not wish to generate graphs, unchecking this box will slightly improve performance.

Table Settings

This dialog allows you to specify how to identify and present information about individual lines. The result is displayed in the table window.

Match with known lines and delta f

If you select this option, for each line in the table the known lines database will be queried for matches within the specified delta f (frequency in hz). These matches will be displayed in the table window.

Match with coherence db

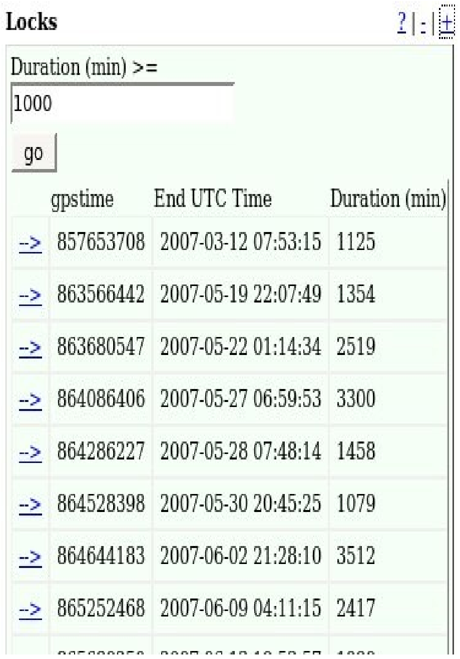
If you select this option, for each line in the table the coherence database (which stores coherences between environmental channels and the dark fringe which are above a certain threshold) will be queried for matches during locks that are within the time range you selected with the data selection dialog. For each channel that has matches, the frequency and coherence of the maximum value in the range will be displayed in the table window. The name of the channel will be a link to open a window to display the complete coherence plot for the selected lock and channel.

Automatic Line Detection

When checked, the web interface will use a tree-search algorithm to find lines with the desired detection resolution and minimum persistency. This search is not 100% reliable and works best for lines that are not wider and do not drift more than the specified resolution.

Manual Line Specification

When checked, the web interface will display entries in the table for the lines you specify in a comma separates list of the form "[min freq]-[max freq], ...". For example: "55-56, 59-59.5". Only lines you specifed that are within (non-inclusive) the selected data will be in the table. I.E, if you have selected 50-60hz, a line '50-51' will not be displayed, but '50.1-51' will.



Locks ? | - | +

Duration (min) >=

1000

go

	gpstime	End UTC Time	Duration (min)
->	857653708	2007-03-12 07:53:15	1125
->	863566442	2007-05-19 22:07:49	1354
->	863680547	2007-05-22 01:14:34	2519
->	864086406	2007-05-27 06:59:53	3300
->	864286227	2007-05-28 07:48:14	1458
->	864528398	2007-05-30 20:45:25	1079
->	864644183	2007-06-02 21:28:10	3512
->	865252468	2007-06-09 04:11:15	2417

Locks

The locks dialog displays a list of science mode locks in chronological order. The locks dialog is minimized by default because of its length, and must be opened by clicking the '+' link next to the title.

Min duration

Only locks with a duration greater than this value in minutes will be displayed in the list.

Use

Clicking the "use" link next to a lock will fill in the "Start Time" and "duration" fields of the data selection dialog with the approriate values for that lock.

Graph

The graph can be saved to your computer by right-clicking and selecting "save-as".

Download Raw Data

Click the link in this window to download the raw data in tab separated value format. This format can be opened by most graphing and analysis programs.

Table

This window displays identified lines and can also display known line and coherence matching information.

APPENDIX B: Common Maintenance and Troubleshooting

Complete Reinstall

In the event that a complete reinstall of the application is necessary, follow these steps:

1) Install an apache server with PHP5 installed as a module with GD2 support.
2) Install a MySQL 4.1 server with support for the InnoDB storage engine enabled.
3) Configure the servers in accordance with the section "Important Server Settings"
4) Create a database for the application to use.
5) Create a user for the application that has privileges to update, select, create, and drop in the database.
6) Create the necessary tables using the SQL script /virgoDev/LineMonitor/web/{version}/scripts/sql/createTables.sql
7) Update the database connection info in the configuration file for LineMonitor.exe and configure it to process all old data that should be in the database.
8) Follow the instructions in "Running LineMonitor.exe On-line" to start the on-line process.
9) Copy the web interface directory /virgoDev/LineMonitor/web/{version}/ to the web servers path
10) Follow the instructions in "Changing server connections for the web interface" to configure the web interface for the new users.
11) CRON the script /virgoDev/LineMonitor/web/{version}/scripts/bash/removeOldImgs.sh to run on a daily basis to clean the temporary images directory of the web server.

Migrating Tables

When migrating line monitor tables to a different server, the only tables that need be moved are 'LM_buffer_manager' and 'LM_points'. Any other tables prefixed with 'LM_' are temporary tables that can be deleted.

Important Server Settings

- PHP5
  - Must have GD2 module.
  - Must have MySQL client module

  - Problems may occur if session id URL writing is not enabled.
  - Session expiration time should be at least twenty minutes.
  - Should have a script memory limit reasonable for the amount of RAM on the machine to prevent errors from taking down the whole system.
- MySQL
  - Heap table maximum size should be set to at least 16mb.
  - Connection timeout should be at least ten minutes to prevent LineMonitor.exe from timing out.
  - InnoDB storage engine must be enabled.

Running LineMonitor.exe On-line

When starting or restarting LineMonitor.exe processing on-line data, it is important to follow these steps to avoid gaps in the data and duplicate data:

1. Connect with the command line mysql client to the LineMonitor database and execute:
   ```
   SELECT MAX(gpstime) FROM LM_points;
   ```
2. Start LineMonitor.exe On-line with the correct configuration file (presumably a modified version of /virgoDev/LineMonitor/{version}/lm_default.cfg). Note the gpstime at which it starts.
3. Run another process of LineMonitor.exe configured to process between the times noted in step one and two.

Changing server connections for the web interface

The file containing server connection information for the web interface is found in the base directory of the interface and is called “lmConnections.inc.php.” Change the necessary connection parameters in this file.

Slow access and “Using Disk” messages

The web interface creates memory tables on the MySQL server to buffer data. If there is a configuration or session error, or if the selected data range is too large, it may not be able to create these tables. This will significantly slow performance, especially if automatic line detection is enabled. If this error is occurring for reasonably sized queries ,check settings with “Important Server Settings”. Also try following the resolution for “Blank graphs / Empty tables”.

Blank graphs / Empty tables

If a query that should have results comes up with a blank graph or an empty table, it probably indicates something has gone awry with the caching system. Execute this command on the MySQL command line client connected to the LineMonitor database to reset the buffer system:

```
DELETE FROM LM_buffer_manager;
```

Then, drop any tables prefixed by 'LM_MEMBUFFER_'.

Graph Images Not Found

Whichever user the apache process is running under needs read, write, and execute access to the directory called tmpimg in the line monitor web interface root directory. Without this, it will not be able to save graphs to image files for display.

APPENDIX C: Web Interface – Detailed Development Reference

File Descriptions:

./chart	This directory contains the include files and examples for the Chart library used to produce graphs.
./help	This directory contains help.html and help.css which contain the users guide.
./index.php	This file is the main workhorse. It includes other files, draws the gui, and creates data sources.
./coherence.php	This file draws a graph of coherence values for one channel and lock and is linked to from index.php.
./lmDownload.php	This file returns data in tab separated value format for downloading and is linked to from index.php.
./lmCleanup.inc.php	This file closes database connections and performs other maintenance
./lmConnections.inc.php	This file contains the connection strings for the different databases used.
./lmData.inc.php	This file contains class definitions for working with data.
./lmGui.inc.php	This file contains class definitions for defining the gui.
./lmLines.inc.php	This file contains class definitions for identifying, matching, and displaying line tables.
./lmOutputGraph.inc.php	This file contains class definitions for drawing graphs.
./lmGui.js	This file contains javascript functions used by the gui.
./lmStyle.css	This file contains the style definitions that determine the appearance and organization of the gui.

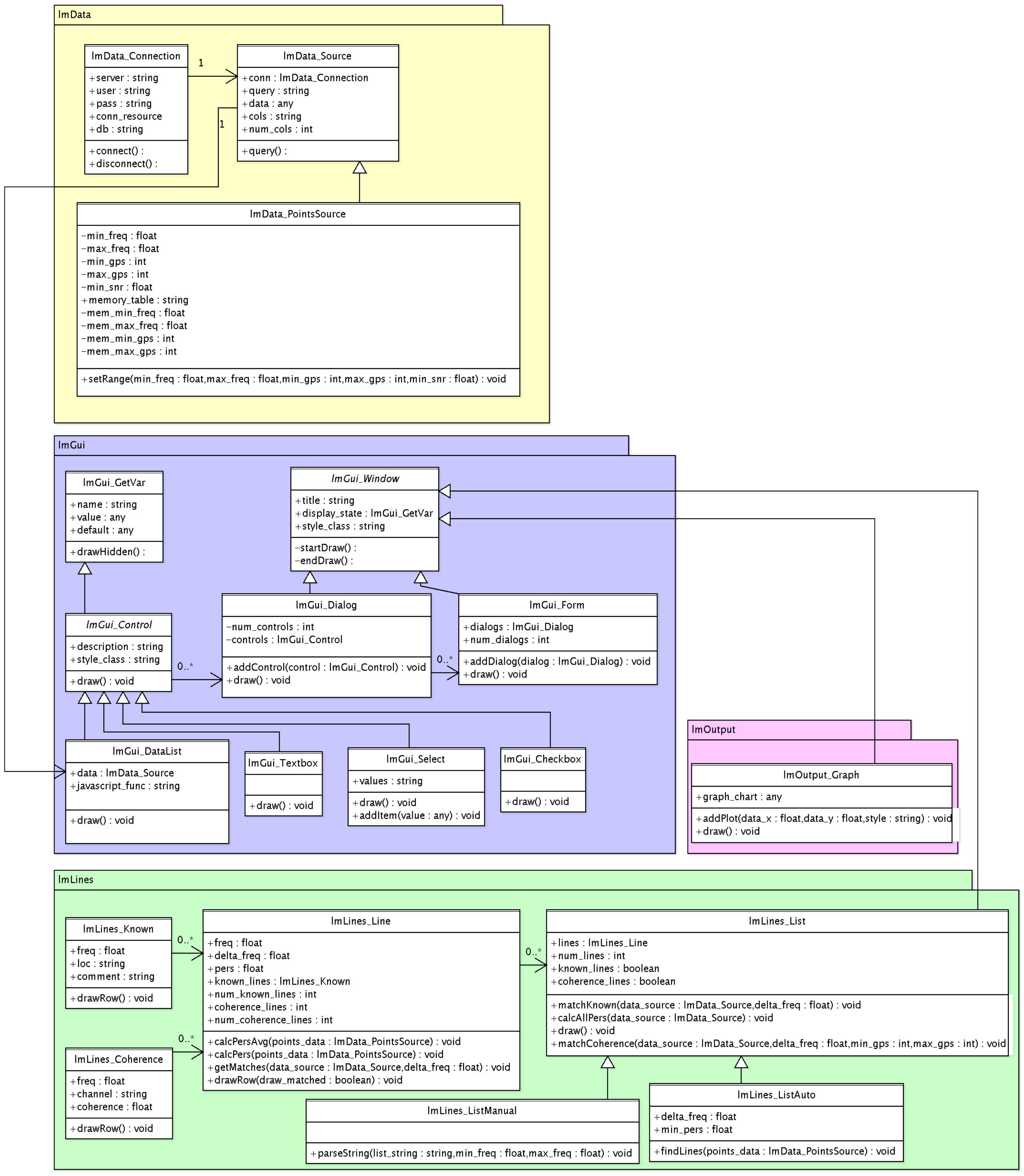
UML Diagrams

The class diagram describes completely the organization and relationships in the code. Different packages correspond to different .inc.php files.

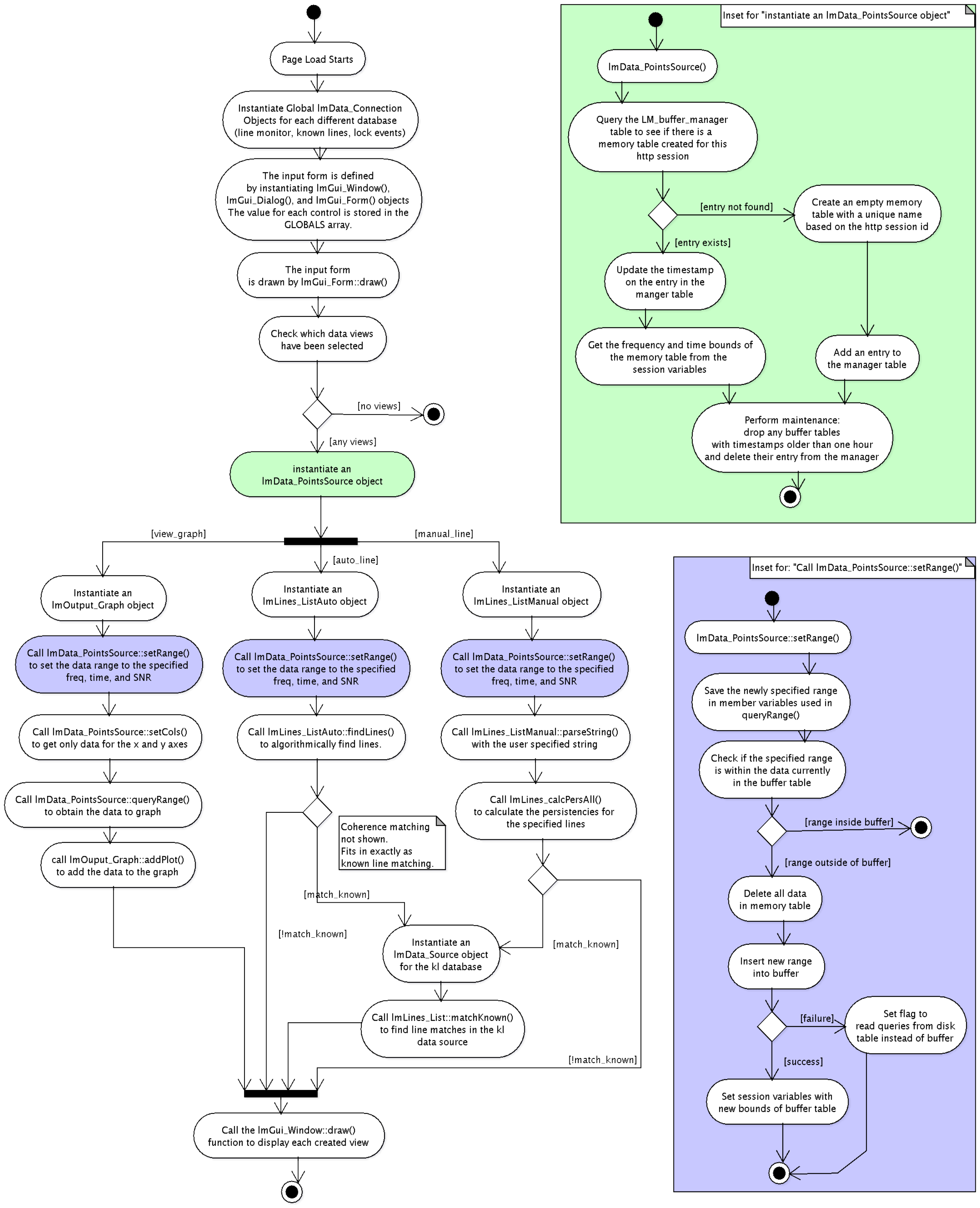
The flow chart describes the orderin which code executes and the general ways tasks are accomplished when index.php is loaded.

The diagrams were created with the open source java software ArgoUML. Their raw files can be found in the docs folder and they should be updated to reflect code changes.

Web Interface Class Diagram



Web Interface Flow Chart


Page Load Starts
Instantiate Global ImData_Connection Objects for each different database (line monitor, known lines, lock events)
The input form is defined by instantiating ImGui_Window(), ImGui_Dialog(), and ImGui_Form() objects The value for each control is stored in the GLOBALS array.
The input form is drawn by ImGui_Form::draw()
Check which data views have been selected
[no views]
[any views]
instantiate an ImData_PointsSource object
[view_graph]
[auto_line]
[manual_line]
Instantiate an ImOutput_Graph object
Call ImData_PointsSource::setRange() to set the data range to the specified freq, time, and SNR
Call ImData_PointsSource::setCols() to get only data for the x and y axes
Call ImData_PointsSource::queryRange() to obtain the data to graph
call ImOuput_Graph::addPlot() to add the data to the graph
Instantiate an ImLines_ListAuto object
Call ImData_PointsSource::setRange() to set the data range to the specified freq, time, and SNR
Call ImLines_ListAuto::findLines() to algorithmically find lines.
Coherence matching not shown. Fits in exactly as known line matching.
[match_known]
[!match_known]
Instantiate an ImLines_ListManual object
Call ImData_PointsSource::setRange() to set the data range to the specified freq, time, and SNR
Call ImLines_ListManual::parseString() with the user specified string
Call ImLines_calcPersAll() to calculate the persistencies for the specified lines
[match_known]
[!match_known]
Instantiate an ImData_Source object for the kl database
Call ImLines_List::matchKnown() to find line matches in the kl data source
Call the ImGui_Window::draw() function to display each created view
Inset for "instantiate an ImData_PointsSource object"
ImData_PointsSource()
Query the LM_buffer_manager table to see if there is a memory table created for this http session
[entry not found]
Create an empty memory table with a unique name based on the http session id
[entry exists]
Update the timestamp on the entry in the manger table
Get the frequency and time bounds of the memory table from the session variables
Add an entry to the manager table
Perform maintenance: drop any buffer tables with timestamps older than one hour and delete their entry from the manager
Inset for: "Call ImData_PointsSource::setRange()"
ImData_PointsSource::setRange()
Save the newly specified range in member variables used in queryRange()
Check if the specified range is within the data currently in the buffer table
[range inside buffer]
[range outside of buffer]
Delete all data in memory table
Insert new range into buffer
[failure]
Set flag to read queries from disk table instead of buffer
[success]
Set session variables with new bounds of buffer table

APPENDIX D: Database Benchmarking

Purpose:

To determine the performance of several different indexes and storage engines against a range of queries and while storing different amounts of data.

Test Variables:

Queries:

Queries were created that selected all fields with different frequency and time ranges based on this matrix. The ranges were determined based on the expected min, max, and averages.

	One Day (86400s)	One Week (604800s)	Two Weeks (1209600s)
2hz	01	05	09
200hz	02	06	10
500hz	03	07	11
2000hz	04	08	12

Data:

The tables were tested once with approximately 30 days of VSR1 data, and once with these 30 days repeated with appropriately incremented IDs and gps times for a total of ½ year of data.

Indexes:

Indexes tested were:

- 01 - a primary key on pointID
- 02 - a natural primary key on (gpstime, freq) [no pointID field]
- 03 - a primary key on pointID and an index on (gpstime)

Storage Engines

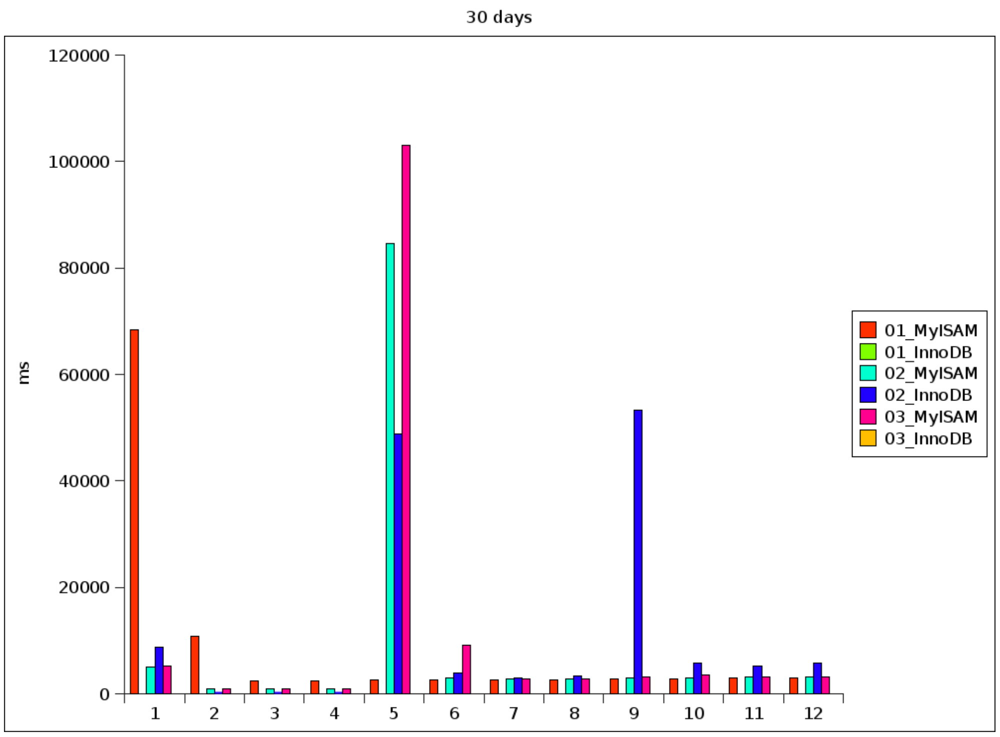
The storage engines tested were:

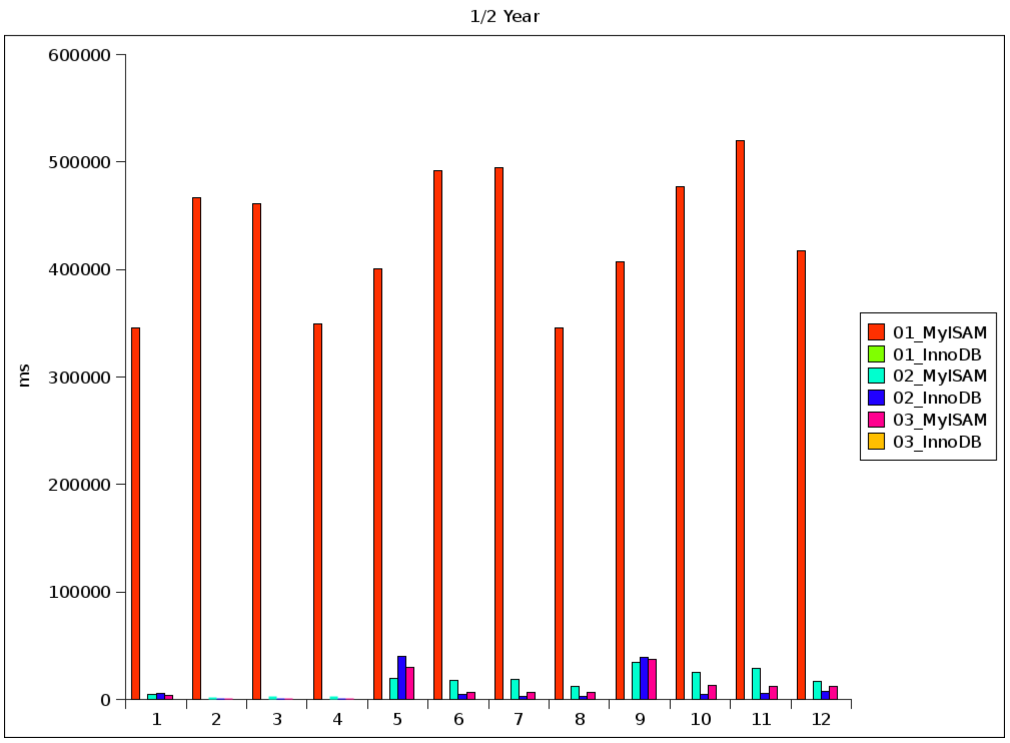
- MyISAM – the defaut MySQL storage engine
- InnoDB

Tools Used

The benchmarking software used was MySQL Super Smacker, combined with shell scripts to generate queries and tables from combinations of the variables.

Results:





Anomalies:

Tables 01_InnoDB and 03_InnoDB were not included in the results because initial attempts took orders of magnitude longer with these tables than others.

Queries one, five, and nine were much slower than any others, especially considering they are the smallest frequency ranges. Caching within the MySQL server was disabled for benchmarking, but it is possible that some system or device cache was affecting the results. Because these queries are all at the start of a new time range, they would not benefit as much as the others from this cache.

Conclusions:

The forerunners of the benchmarks are tables 02_InnoDB and 03_MyISAM. The general trend seems to be that MyISAM performs better with only 30 days of data in the table and InnoDB better with ½ a year. The explain statements on the queries show that InnoDB always performs a range scan on its primary key, while MyISAM performs full table scans on the 30 day table and range scans on the gpstime index. The performance difference makes sense because InnoDB stores data in order of the primary key, and is therefore able to do quick range scans on the primary key. MyISAM, on the other hand, is heavily optimized for full table scans or single index queries. Because of the way InnoDB works, we expect its performance to stay constant for range scans no matter how much more data is in the table. MyISAM, on the other hand, will only continue to get worse as each single index query takes longer. Additionally, although it was too complicated to test in this setup InnoDB has the advantage of row level locking vs. MyISAM's table locking, which should speed up queries when LineMonitor.exe is writing to the table.