A Comprehensive System for Non-Intrusive Load Monitoring and Diagnostics

by

James Paris

S.B., Massachusetts Institute of Technology (2003)
M.Eng., Massachusetts Institute of Technology (2006)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2013

Author .....
Department of Electrical Engineering and Computer Science
August 30, 2013

Certified by .....
Steven B. Leeb
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Committee on Graduate Students

A Comprehensive System for Non-Intrusive Load Monitoring and Diagnostics

by

James Paris

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Energy monitoring and smart grid applications have rapidly developed into a multi-billion dollar market. The continued growth and utility of monitoring technologies is predicated upon the ability to economically extract actionable information from acquired data streams. One of the largest roadblocks to effective analytics arises from the disparities of scale inherent in all aspects of data collection and processing. Managing these multifaceted dynamic range issues is crucial to the success of load monitoring and smart grid technology.

This thesis presents NilmDB, a comprehensive framework for energy monitoring applications. The NilmDB management system is a network-enabled database that supports efficient storage, retrieval, and processing of vast, timestamped data sets. It allows a flexible and powerful separation between on-site, high-bandwidth processing operations and off-site, low-bandwidth control and visualization. Specific analysis can be performed as data is acquired, or retroactively as needed, using short filter scripts written in Python and transferred to the monitor.

The NilmDB framework is used to implement a spectral envelope preprocessor, an integral part of many non-intrusive load monitoring workflows that extracts relevant harmonic information and provides significant data reduction. A robust approach to spectral envelope calculation is presented using a 4-parameter sinusoid fit.

A new physically-windowed sensor architecture for improving the dynamic range of non-intrusive data acquisition is also presented and demonstrated. The hardware architecture utilizes digital techniques and physical cancellation to track a large-scale main signal while maintaining the ability to capture small-scale variations.

Thesis Supervisor: Steven B. Leeb
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am extremely thankful to Professor Steven Leeb for his persistent guidance, enduring support, and ceaseless enthusiasm for everything. I wish to also thank Professors Les Norford and James Kirtley for their continued input and assistance, as well as Professor Steven Shaw of Montana State University for creating the foundation for much of this work.

Essential support for this research was graciously provided by the MIT Energy Initiative, the BP-MIT Research Alliance, The Grainger Foundation, the Massachusetts School Board Authority, Cottage Elementary School, and Ken Wertz.

I would also like to extend thanks to my many peers and colleagues, including Rob Cox, Warit Wichakool, Al Avestruz, Chris Laughman, Uzoma Orji, John Cooley, Chris Schantz, BJ Thompson, Arthur Chang, John Rodriguez, Zack Remscrim, Zach Clifford, Mark Gillman, Ariel Rodriguez, and Mariano Alvira. In particular, this work would not have been completed were it not for the inspiration and masterful help of John Donnal.

Finally, I thank my family and friends for their endless and unconditional encouragement, with the utmost thanks and appreciation to Madeleine for her patience and love.

Contents

List of Figures

4 The Sinefit Spectral Envelope Preprocessor

5 Zoom NILM: Physically-Windowed Sensor Architecture

List of Tables

List of Algorithms

List of Code

Chapter 1

Introduction

Energy monitoring and smart grid applications have rapidly developed into a multi-billion dollar worldwide market. Residential, commercial, and industrial systems increasingly rely on electrical and electromechanical monitoring to improve efficiency, diagnose faults, and provide for informed control. The non-intrusive load monitor (NILM) has been shown to be an effective and efficient energy monitoring system, and has been applied to a wide variety of systems [1–20]. The non-intrusive approach seeks to reduce sensor count by performing high-resolution, high-speed measurements of the power usage of aggregate loads. Figure 1-1 shows a typical implementation of a NILM, monitoring all loads connected to a distribution panel using only one current and voltage measurement per phase.

A diverse array of diagnostic and condition-based maintenance indicators have been developed for the NILM. These include vacuum system leak detection [8, 21, 22], motor speed estimation based on principal slot harmonics [23], motor coupling failure [22, 24], compressor short-cycling and liquid accumulation [25, 26], fan imbalance and vibration detection [27], and more. Many of these indicators require non-trivial computation, such as frequency-domain analysis, simulation, and long-term state tracking, and rely on high-speed data rates of up to 8 KHz sampling per channel. As power distribution networks are increasingly pressed into "dual-use" service, combining power distribution with diagnostic and monitoring capabilities, the desire to apply NILM technology to a wider variety of systems is quickly growing.

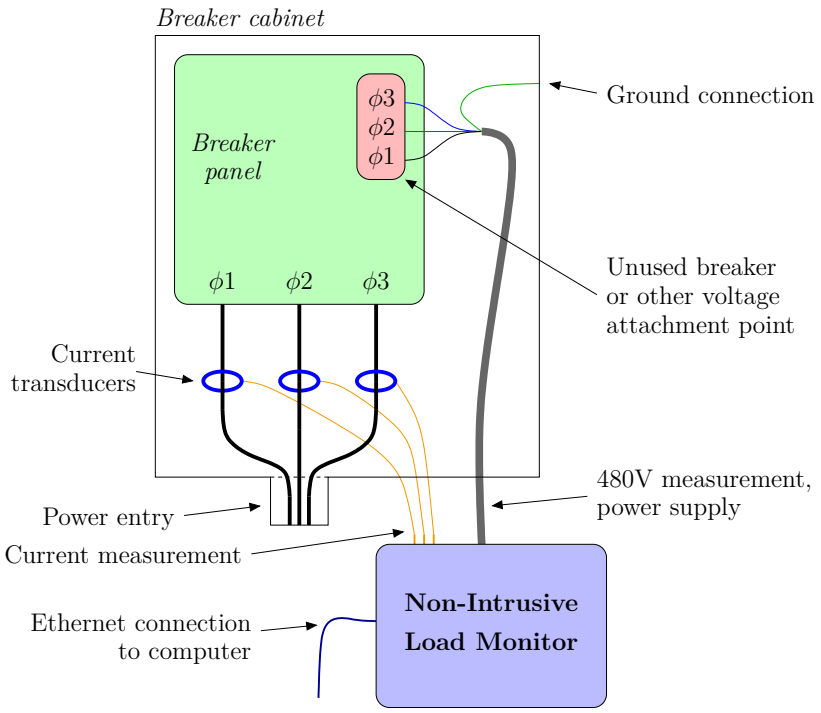


Figure 1-1: Typical installation of the non-intrusive load monitor. Current and voltage are measured, providing the NILM with detailed power usage data for all loads connected to the distribution panel.

This thesis provides solutions to several problems with power monitoring for energy scorekeeping and diagnostics. It presents a framework that rethinks the distribution of computation in a monitoring network, easing access and interpretation of data, while minimizing the use of expensive or limited resources like network bandwidth. It introduces new signal processing that exploits the physical interpretation of observed waveforms to extract useful information, while offering compression and reduced storage requirements. This thesis also introduces new hardware for data acquisition, which offers improved dynamic range by exploiting the time scale separation inherent in many utility waveforms.

1.1 "Big Data" Analytics

Monitoring technologies are useful when they can economically extract actionable information from acquired data streams. Smart grid deployments have generally focused on instrumentation and communications infrastructures, and are only now

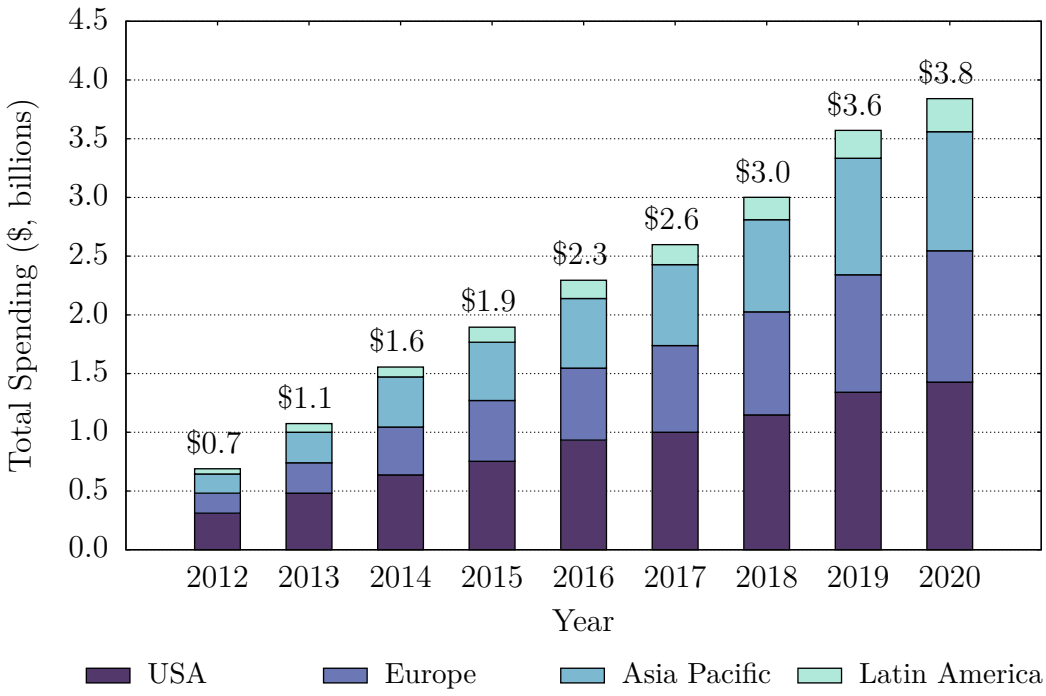


Figure 1-2: Predicted yearly spending on smart grid data analytics. 2013-2020. Cumulative worldwide spending on smart grid "big data" will exceed $20 billion by 2020. This is in addition to existing spending on communications, monitoring, and infrastructure layers. From [28].

beginning to shift focus to the tools and processes for analyzing and making use of the captured data [28]. Expenditures on data analytics are predicted to grow significantly through the end of the decade, as shown in Figure 1-2.

One of the largest roadblocks to effective analytics arises from the disparities of scale inherent in all aspects of data collection and processing. For example, relevant electrical information can fall within a huge range of frequency and amplitude. Specific diagnostic indicators may require millisecond sampling, monthly statistics, or both. Sensors can usefully capture information orders of magnitude faster than existing networks can transfer it. Managing these multifaceted dynamic range issues is crucial to the success of load monitoring and smart grid technology.

Several high-profile attempts and failures have demonstrated the difficulty of managing the scale of non-intrusive load monitoring. Google PowerMeter and Microsoft Hohm, both introduced in 2009, were intended to help homeowners reduce power usage by providing power usage data from the utility supplier with a convenient

web-based interface. Each was discontinued within two years, citing poor uptake by consumers [29,30], despite pilot studies in the same time frame demonstrating strong consumer interest in home monitoring [31]. Some users cited a lack of real-time data, with a lag of one or more days, and an inability to disaggregate and identify individual loads as some of the weaknesses of the system [32].

1.2 Contributions and Organization

This thesis focuses on building a complete solution to the NILM data analytics problem by addressing the challenges posed by dynamic range and data scale. Chapter 2 presents NilmDB, a comprehensive framework for energy monitoring applications. The NilmDB management system is a network-enabled database that supports efficient storage, retrieval, and processing of vast, time-stamped data sets. It allows a flexible and powerful separation between on-site, high-bandwidth processing operations and off-site, low-bandwidth control and visualization. Specific analysis can be performed as data is acquired, or retroactively as needed, using short filter scripts written in Python and transferred to the monitor. The system is complemented by NILM Manager, a fluid, web-based user interface that provides an unprecedented level of access to and control of the load monitoring process.

The NilmDB framework is used in Chapter 4 to implement a spectral envelope preprocessor, an integral part of many non-intrusive load monitoring workflows that extracts relevant harmonic information and provides significant data reduction. A robust signal processing approach to spectral envelope calculation is presented using a 4-parameter sinusoid fit, with advantages for noisy and variable-frequency loads. A mathematical framework for determining the resolution and accuracy of spectral envelope output is presented, allowing more efficient design of non-intrusive sensor and acquisition systems.

Finally, in Chapter 5, a new physically-windowed sensor architecture for improving the dynamic range of a data acquisition system is introduced. This hardware architecture utilizes digital techniques and physical cancellation to track a large-scale

main signal while maintaining the ability to capture small-scale variations. This is particularly applicable to non-intrusive, condition-based maintenance applications such as motor speed estimation from principal slot harmonics. Its performance is demonstrated in a prototype current measurement system based on a closed-loop Hall sensor.

Chapter 2

NilmDB

2.1 Introduction

The NILM Database (NilmDB) is a comprehensive framework designed to solve the "big data" problem of non-intrusive load monitoring and diagnostics. It provides the central component of a flexible, distributed architecture for the storage, transfer, manipulation, and analysis of time-series data. NilmDB is network-transparent and facilitates remote viewing and management of large data sets by utilizing efficient data reduction and indexing techniques. By leveraging industry-standard protocols and software tools, the server supports advanced encryption, authentication, and authorization controls.

At its core, NilmDB is a high-performance database that stores and indexes time-stamped streams of data. It defines a structured application programming interface (API) with commands to insert, extract, filter, and process this data. A wide variety of tools that utilize this API are provided as standard components and libraries under the NilmDB umbrella. The system is further supported by the NilmRun server, which enables the execution of filters and tools on remote or distributed systems. Together, these constitute a complete framework for NILM data analytics and management.

NilmDB was developed in conjunction with NILM Manager by John Donnal [33]. NILM Manager is a unified and centralized management interface to the distributed NilmDB system. It utilizes the NilmDB and NilmRun APIs to provide end-users

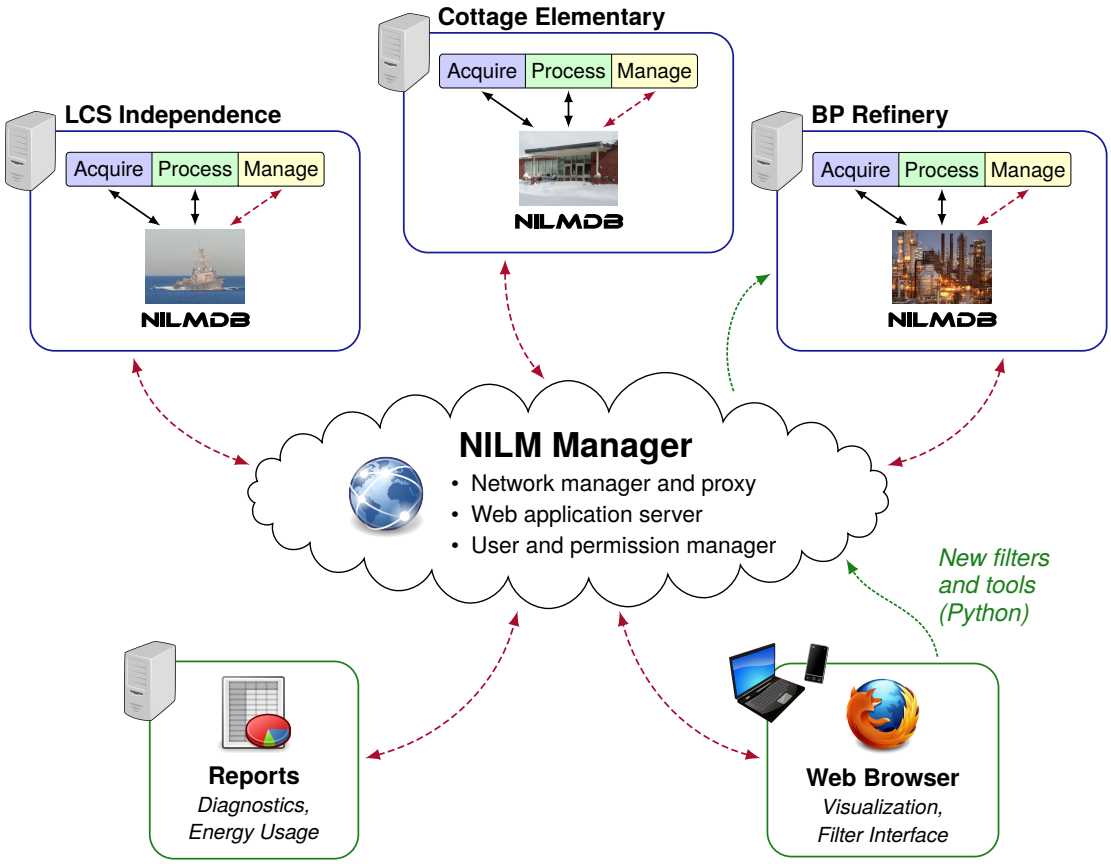


Figure 2-1: NilmDB network and global system architecture. Individual NilmDB instances are accessed and controlled through NILM Manager. Solid lines represent high-bandwidth data transfer pathways, and dashed lines represent efficient low-bandwidth connections.

with a powerful, web-based data visualization and manipulation platform. Data is retrieved dynamically from a NilmDB backend as needed, allowing a remote user to seamlessly view data on timescales ranging from decades to microseconds on even low-bandwidth connections. Process management moves computation to remote nodes, allowing dynamic testing and reconfiguration of filters, diagnostics, and reports. NILM Manager acts as a proxy and connection broker, allowing controlled user access to a secure NilmDB network from any location.

Figure 2-1 demonstrates the global system architecture of the NilmDB framework. In general, one or more unique instances of NilmDB are installed at a monitored site. Each NilmDB instance manages and stores data local to that particular system or network. These instances are then accessed globally through the NILM Manager. Thus,

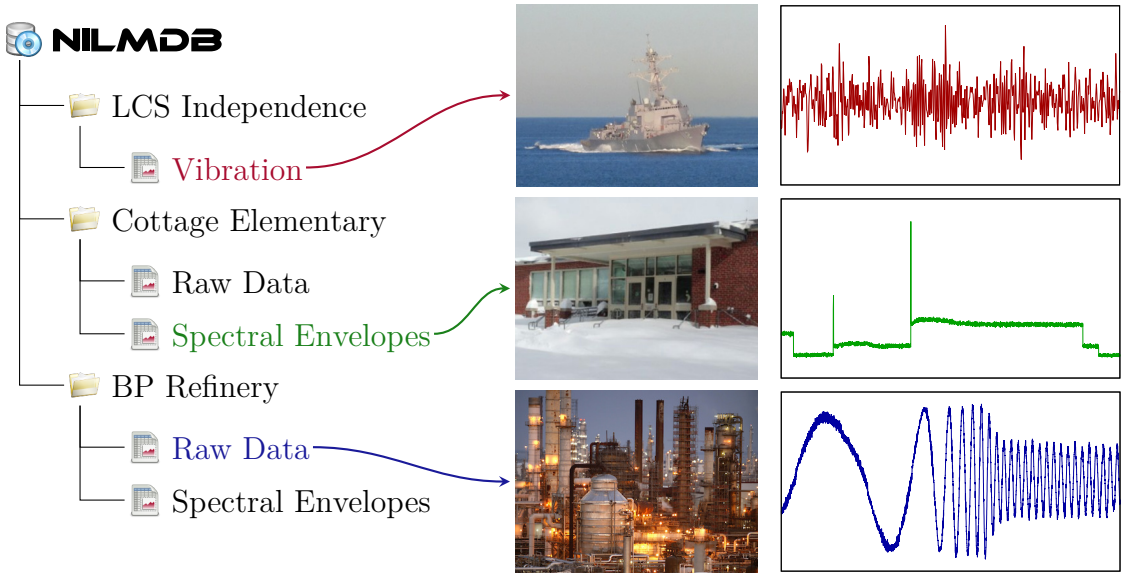
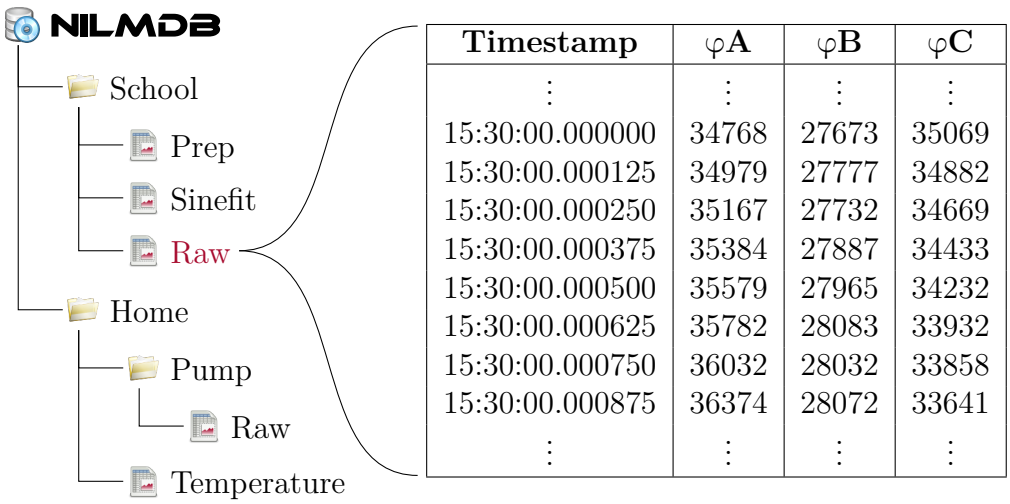


Figure 2-2: NilmDB stream examples. Here, a single NilmDB instance contains five streams with data for three different sites, organized in a tree-like structure.

high-bandwidth storage and processing of large quantities of data can be restricted to the local NilmDB systems. While end-user systems always have the option of retrieving full data where network bandwidth permits, this system enables the user to execute filters and tools on the remote systems that reduce the data into more manageable sizes. Such data can then be retrieved for visualization, reporting, or diagnostic purposes.

2.2 System Concepts

The NILM database framework is structured as a client-server architecture. The server process, called simply NilmDB, runs on a single system and stores data on that system. Client programs can run on the same system or any other system, and connect to this server to perform actions such as inserting new data and extracting existing data. The following sections introduce the basic concepts of NilmDB data storage and processing.



Timestamp	φA	φB	φC
⋮	⋮	⋮	⋮
15:30:00.000000	34768	27673	35069
15:30:00.000125	34979	27777	34882
15:30:00.000250	35167	27732	34669
15:30:00.000375	35384	27887	34433
15:30:00.000500	35579	27965	34232
15:30:00.000625	35782	28083	33932
15:30:00.000750	36032	28032	33858
15:30:00.000875	36374	28072	33641
⋮	⋮	⋮	⋮

Figure 2-3: Data contained within a NilmDB stream. A stream contains a fixed number of columns of a homogeneous type, and can be conceptually viewed as a table with an unbounded number of rows. Each row holds a single unique timestamp and the data for that time.

2.2.1 Streams and Data

All data in NilmDB is organized within streams. Streams contain homogeneous time-series data from a particular source. This data can represent physical quantities, computed values, or any other timestamped information. Examples include voltage, current, temperature, vibration, spectral envelopes, system run-time and health metrics, error and event indicators, etc. Streams are organized and identified in the database using a tree-like path structure that mirrors an arrangement of files and folders.

Figure 2-2 demonstrates examples of NilmDB streams within this structure. The paths are denoted using the components of the tree structure separated by "`/`". For the shown example, paths might be referenced as "`/LCS Independence/Vibration`" or "`/BP Refinery/Raw Data`". In practice, names without spaces are generally preferred as they are easier to work with when using command-line tools.

Streams can be viewed conceptually as large tables of data, as shown in Figure 2-3. Each row contains a unique timestamp and data that matches the stream's layout, which is determined when the stream is created and indicates the number of columns and their data type. The data types and layouts are further described in Section 2.2.3.

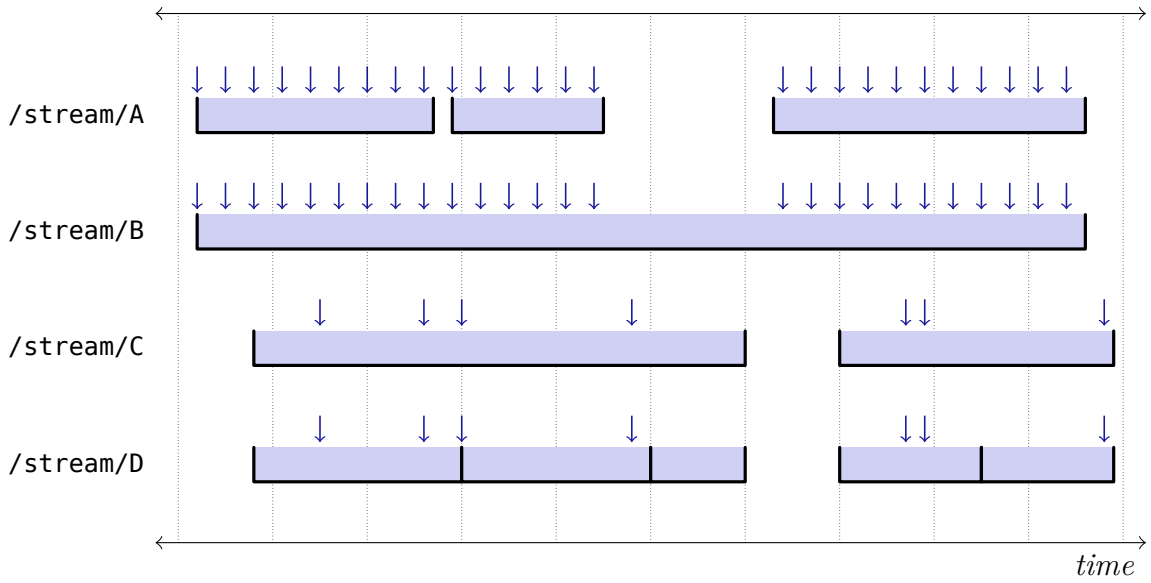


Figure 2-4: NilmDB stream interval examples. The underscored rectangles indicate the intervals in a stream, and the arrows point to specific data timestamps within those intervals. Streams A and B contain the same data, but do not cover the same time intervals; conversely, streams C and D are functionally equivalent.

2.2.2 Intervals

Streams track data in non-overlapping intervals. Each interval denotes a start time S and an end time E and is half-open; that is, interval $[S \to E\rangle$ contains data points with timestamps t satisfying:

$$S \leq t < E \tag{2.1}$$

All rows of data are inserted into the database with associated intervals. The number of data points that may be in a single interval is limited only by the resolution of the data timestamps, which must be unique. Intervals may also be sparse or empty. This could be used by an event identifier, for example, to indicate a region of time for which data was processed, but no matching events were detected.

Figure 2-4 shows examples of intervals and their data for four different streams. Intervals are denoted by the underscored rectangles, and arrows represent the timestamps of specific rows of data within those intervals. Note that intervals may include timestamps corresponding to their start, but not their end, and that two streams can

differ based on their intervals even if they contain the exact same data. Similarly, two contiguous intervals $[T_1 \to T_2\rangle$ and $[T_2 \to T_3\rangle$ are functionally equivalent to the one long interval $[T_1 \to T_3\rangle$.

Intervals are immutable. Creating an interval and inserting data into that interval is a combined operation, as described in Section 3.2.1.13. Once created, no further data can be added to, or removed from, that particular interval. Instead, new non-overlapping intervals can be created in the same manner, or intervals or segments thereof can be removed together with their data. In the example intervals shown in Figure 2-4, the practical difference between `/stream/A` and `/stream/B` is that additional data can be added in the gap in the former, but not the latter. Either stream could be transformed into the other, by adding new empty intervals to `/stream/A` or by removing specific ranges from `/stream/B`.

Client operations on the stream, such as extracting or removing data, can specify a particular interval of time in their request, $[S_R \to E_R\rangle$. The database satisfies these requests by computing the intersections between the request and all database intervals $[S_n \to E_n\rangle$:

$$I = \bigcup ([S_R \to E_R\rangle \cap [S_n \to E_n\rangle) \tag{2.2}$$

The server then acts on all data corresponding to the intervals in I. Creating the set I may involve internally splitting or truncating intervals, which is generally transparent to the user. This operation is one of the key features of NilmDB, as it allows the client applications to freely work with data intervals that need not correspond to the intervals created at insertion.

2.2.3 Data Types and Timestamps

Streams are created with a specific layout, which describes the type and number of data values within the stream. The layout is described with a string of the form "`type_count`". Each row of data contains one timestamp and exactly `count` values of type `type`. Table 2-1 lists all of the supported types and the values that they can

Type	Description	Supported Values
`int8`	8-bit signed integer	$-128 \rightarrow 127$
`uint8`	8-bit unsigned integer	$0 \rightarrow 255$
`int16`	16-bit signed integer	$-32{,}768 \rightarrow 32{,}767$
`uint16`	16-bit unsigned integer	$0 \rightarrow 65{,}535$
`int32`	32-bit signed integer	$-2{,}147{,}483{,}648 \rightarrow 2{,}147{,}483{,}647$
`uint32`	32-bit unsigned integer	$0 \rightarrow 4{,}294{,}967{,}295$
`int64`	64-bit signed integer	$-2^{63} \rightarrow (2^{63}-1)$
`uint64`	64-bit unsigned integer	$0 \rightarrow (2^{64}-1)$
`float32`	Single-precision floating point	$\pm 10^{38}$, ~ 7 significant digits
`float64`	Double-precision floating point	$\pm 10^{308}$, ~ 16 significant digits

Table 2-1: Supported stream data types and ranges. The number in the type represents the bits of storage used; lower values result in smaller data. Floating point values approximate real numbers and follow the IEEE 754 specification [34].

store. For example, a layout of `uint16_6` means that a stream stores six non-negative integers in the range $0 \rightarrow 65535$, plus one timestamp, per row. The `count` is an integer between 1 and 1024. It is generally expected that counts will be in a lower range such as 1 to 32, and the database is optimized for this range.

Timestamps and interval endpoints in NilmDB are always stored as `int64` values, regardless of the layout of the stream. NilmDB uses timestamps for indexing and searching, but otherwise applies no specific interpretation to these values. However, many of the NilmDB client tools and software, including the NILM Manager, follow the convention of interpreting this timestamp as the integer number of microseconds elapsed since the "Unix epoch" of January 1, 1970 at 00:00:00 UTC[1].

Some software languages that may interact with NilmDB, such as Javascript, internally force the use of IEEE 754 double-precision floating-point numbers, and as such, can only exactly represent integers in the range $\pm 2^{53}$. Thus, limiting timestamps to this range increases compatibility with such systems. Even with this limitation, the conventional interpretation of timestamps can represent dates until the year 2255.

[1]To be precise, NilmDB timestamp t is interpreted as "Unix time" $t/10^6$, which differs from the simpler description in that leap seconds are not included in Unix time.

2.2.4 Metadata

Besides timestamped data, NilmDB also supports storing metadata with any number of arbitrary key/value pairs for each stream. Keys and values are both textual strings. Within a particular stream, keys must be unique. These key/value pairs can be used for any ancillary information that should be stored alongside the stream. For example, a "`scale_factor`" key with a value of "`1.337`" might be used to indicate a conversion ratio. Many of the NilmDB filters described in Section 3.4.2 use metadata to ensure that their parameters are the same between multiple invocations of the filter.

More complex data types can be stored in metadata by first encoding them as strings. For example, the NILM Manager in Section 2.4 makes extensive use of stream metadata by converting complex data structures to Javascript Object Notion (JSON) format before storing values [35]. The length of metadata values is essentially unbounded, but the protocols for setting and retrieving metadata are designed around the assumption that it will be relatively short, on the order of kilobytes.

2.2.5 Filters

Data analysis and processing within the NilmDB framework is accomplished through the use of filters. The NilmDB server provides a limited vocabulary of actions, such as inserting data and extracting data, and filters build upon these actions in order to transform data in a meaningful and customizable way. Filters are clients to the NilmDB server and access the database like any other process; their definition is primarily conceptual and does not enforce any particular implementation. The most basic filters extract data from a "source" stream, perform arbitrary processing and transformation of that data, and re-insert it into a new "destination" stream. For example, the sinusoid-fitting filter `nilm-sinefit` (Section 3.4.2.10) identifies zero crossings in a source stream and marks them in a destination stream.

The interval management of NilmDB allows filters to efficiently process data by considering only those intervals that are present in the source, and not yet present in the destination. Figure 2-5 shows an example of this for a filter with one source and

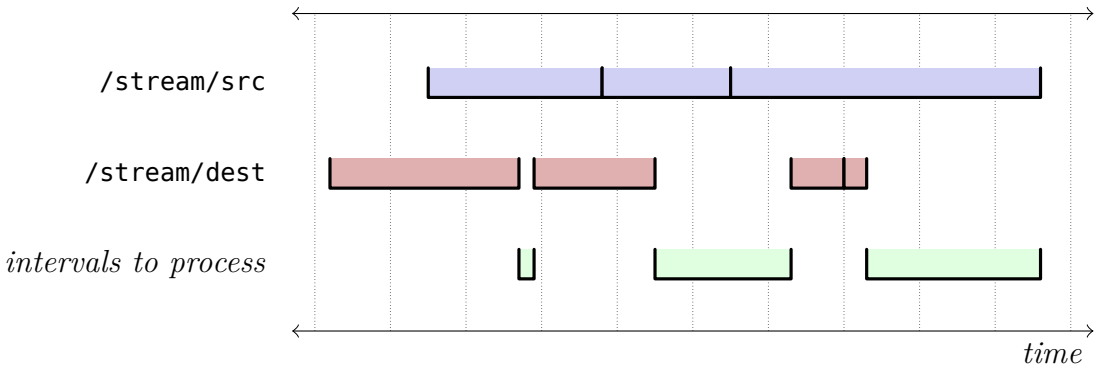


Figure 2-5: Filter processing based on intervals. A filter can use the intervals present in its source and destination streams to efficiently determine which time ranges of data still need to be processed.

one destination. If the filter is interrupted and restarted, or if new data is inserted into the source stream and the filter is run again, this technique allows the filter to easily identify and continue running against the remaining unprocessed data.

Filters can utilize multiple streams for input and output. For example, the spectral envelope calculation in `nilm-prep` (Section 3.4.2.9) uses both raw data and calculated zero crossings as input. The streams also do not necessarily need to be aligned in time; the event matching in `nilm-trainola` (Section 3.4.2.11) utilizes exemplars chosen at time t_1 while attempting to identify loads at a different time t_2.

2.3 Architecture and Implementation

The general architecture of the NilmDB server is shown in Figure 2-6. NilmDB follows a client/server model, where multiple clients can simultaneously access the server and perform requests and actions. The typical structure of clients is shown in the first two rows. Below the line marked (1) is the NilmDB server. The server runs as a single instance on one computer, and manages the incoming requests through a queue that maintains overall database consistency by passing on only one request at a time. Line (2) in the diagram marks this distinction between multi-threaded and single-threaded processing. The lower levels then perform the specified NilmDB action, working with the streams, intervals, and data according to the request.

NilmDB is primarily implemented in Python, with performance-critical sections

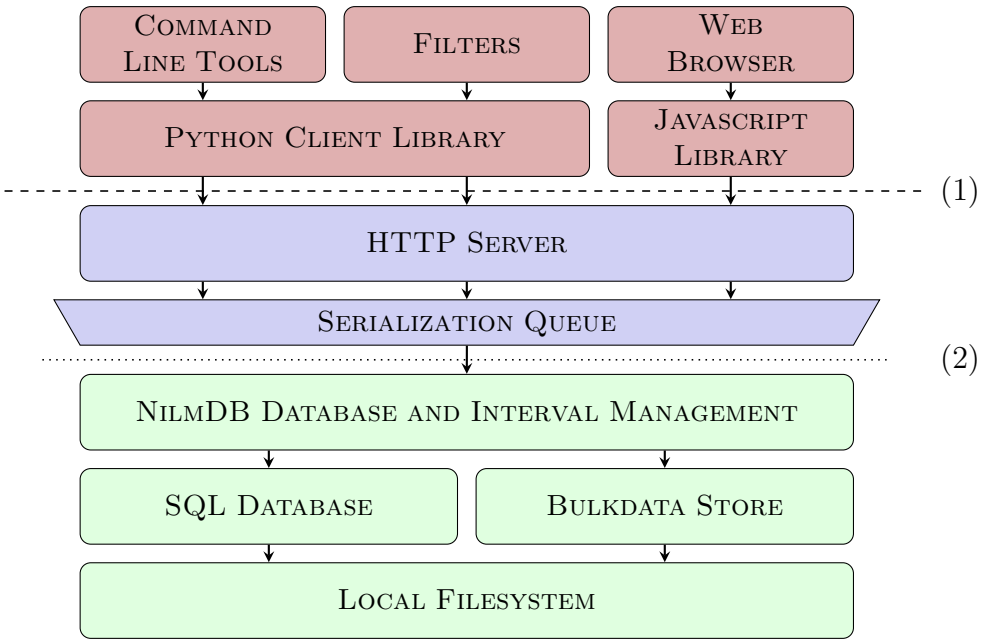


Figure 2-6: NilmDB system architecture and implementation. Client libraries interact with the NilmDB server through an HTTP interface. The server manages storage using a standard SQL database and a custom bulk data store. Line (1) marks the split between clients and server, and line (2) marks the split between multi-threaded and single-threaded processing.

implemented as C extension modules. The following sections describe the various components and their implementation details.

2.3.1 Clients

End users access the server through a variety of user interface clients, such as the command-line interfaces described in Section 3.2.3, or a web application served up by NILM Manager. These applications typically utilize a library layer like the Python client libraries in Section 3.2.2, which abstract away the connection and protocol details of communicating with NilmDB.

Regardless of the source, all interaction with the NilmDB server takes place through a standard HTTP/1.1 compliant interface [36]. HTTP defines methods that perform actions on particular resources, which are identified by Uniform Resource Locator (URL). These actions correspond to the fundamental NilmDB operations, such as creating a stream, listing available intervals, inserting data, extracting data,

etc. The definitions of these actions, as well as their input and output data formats and parameters, form the NilmDB HTTP Application Programming Interface (API).

Complete details about the HTTP API, the client library APIs, and command line interfaces can be found in Chapter 3.

2.3.2 HTTP Server and Serialization

The HTTP server receives and responds to requests from the client. The server implementation is based on the CherryPy web framework [37], which provides tools for developing HTTP-based servers that conform to the Web Server Gateway Interface (WSGI) [38]. Compatibility with WSGI allows NilmDB to be hosted by a full-featured web server, such as Apache, enabling a wide range of features, including encryption, authentication, and compression. The NilmDB source code therefore avoids a significant amount of the complexity of HTTP, and the server primarily acts as an interface between the HTTP API and the underlying NilmDB actions.

The WSGI application interface is inherently multi-threaded, as client connections and requests can come in at any time, and receiving requests or sending responses might proceed slowly, depending on network conditions. Accordingly, the server supports and can transfer data on any number of simultaneous connections. On the other hand, operations that modify the database state generally need to be performed one at a time, in order to maintain consistency. While complex fine-grained locking and ordering may allow some operations to run concurrently, NilmDB uses a more straightforward approach, where direct database access must be performed from one thread only.

Single-threaded access is accomplished through the use of the "serializer" module, which allows any running thread to enqueue a function call "request" and wait for the result. When the database is not busy, the serializer will retrieve the earliest request from the queue, perform its function call in a single, global thread, and return the result to the original thread. Thus, the serializer ensures that any operations on the database are serviced one at a time, in first-in, first-out (FIFO) order. The process of running queued requests in the serializer thread is shown in Figure 2-7.

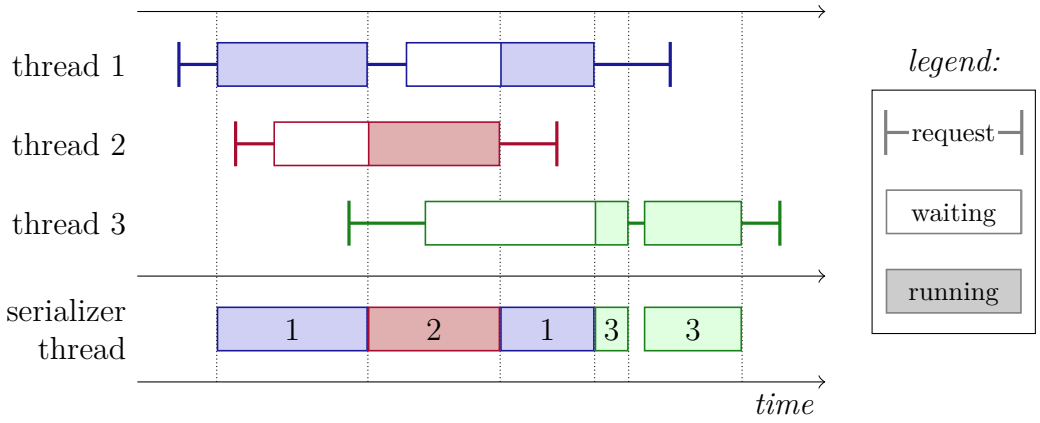


Figure 2-7: Serialization of database operations. Incoming HTTP requests are handled by a multi-threaded server, which performs lower-level database operations through a serializer thread. The serializer runs all operations in a single thread, in the order that they are enqueued, while the threads wait for the their submitted operations to complete.

Some HTTP requests may take an unbounded amount of time to complete, such as extracting data from a large interval. To ensure fairness between clients, the NilmDB database layer may choose to only perform a portion of the requested operation before returning, which gives other threads that are waiting for the serializer a chance to run. The HTTP server handles such occurrences automatically, resubmitting the remainder of the operation to the serializer until it is complete. The client sees it as single HTTP request and response.

2.3.3 Database Management

The NilmDB database layer is responsible for managing the low-level storage and retrieval of streams, intervals, data, and metadata. All data is stored under a directory on the local filesystem. There are two primary data stores:

- A SQL backend, based on the SQLite embedded database library.
- A custom "bulkdata" storage backend, used for stream data contents.

The bulkdata storage acts as an addressable row store; each row of stream data is assigned a sequential unique number ρ and can be later retrieved by that number. It is described in more detail in Section 2.3.4.

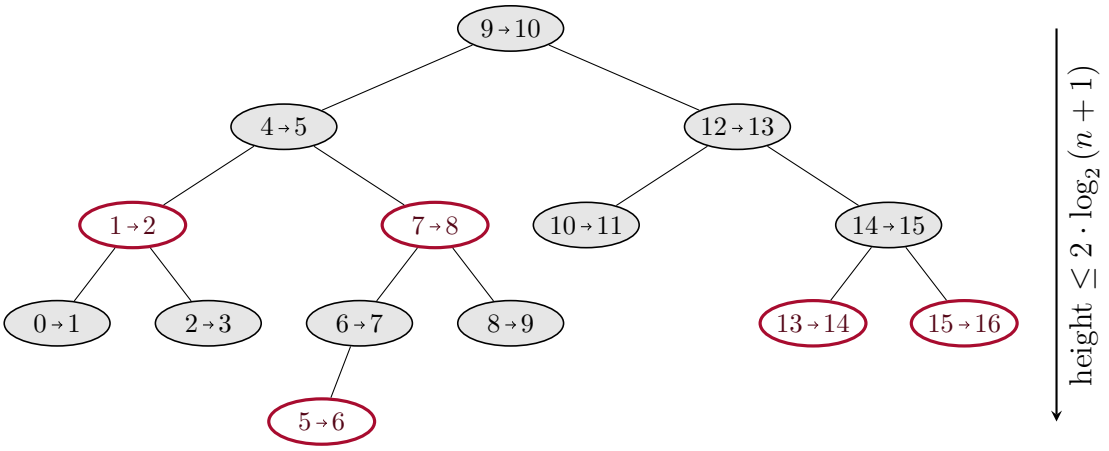


Figure 2-8: Red-black binary search tree for stream intervals, keyed on the interval start time. The tree tracks a per-node "coloring" of red (empty) or black (shaded), which is used to maintain invariants that ensure that the height of a tree with n nodes is at most $2 \cdot \log_2(n+1)$. Here, 14 intervals have been added in random order.

The SQL database stores stream names, layouts, and metadata keys and values. It also contains a list of the intervals for each stream. Specifically, for each time interval $[S \rightarrow E\rangle$, it stores S, E, and the associated bulkdata row numbers ρ_S and ρ_E for those endpoints. The bulkdata storage for any particular interval is contiguous and sorted, and so the rows from ρ_S and ρ_E therefore represent all of the data in that interval.

2.3.3.1 Interval Trees

To maintain performance, NilmDB operations that need to manipulate stream intervals do not interact directly with the SQL database. Instead, an in-memory data structure known as the interval tree is created from the flat list of intervals. These trees are cached for recently-used streams, with changes written back to the SQL database as they are made.

The interval tree is a red-black tree, a form of binary search tree with a per-node "coloring". Figure 2-8 shows an example of such an interval tree for a sequence of 14 contiguous intervals that were inserted in a random order. Each node contains the interval parameters S, E, ρ_S, and ρ_E. The key used for ordering the nodes is the interval start timestamp; since intervals in a stream are non-overlapping, this ordering

is well-defined and unique. For any given node, the left branch contains only earlier intervals, and the right branch contains only later intervals. This property can be used to efficiently search for an interval that includes a given timestamp, by recursively descending the tree. Other operations, such as finding the interval that immediately succeeds a given node, can be implemented by walking the tree in a similar manner.

The coloring of a red-black tree is used to guarantee a balanced structure, ensuring that the longest path is no more than twice the length of the shortest path [39, ch. 13]. This balance is achieved by maintaining the following invariants:

- All nodes are either red or black.
- The root node is black.
- If a node is red, its children are both black.
- All paths from the bottom of the tree to a particular node contain the same number of black nodes.

When inserting or deleting nodes from the tree, these invariants can be maintained in $\mathrm{O}\big(\log_2 n\big)$ time by recoloring and moving nodes as necessary [39]. The maximum height of a balanced tree with n nodes is at most $2 \cdot \log_2 (n + 1)$, and so search operations also take $\mathrm{O}\big(\log_2 n\big)$ time. Thus, the time needed to insert, remove, and locate a specific interval in a stream grows with the logarithm of the number of intervals present.

2.3.3.2 Data Extraction

One of the defining features of NilmDB is the ability to quickly extract data that falls within a specified request interval $[S_R \to E_R\rangle$, independent of the intervals currently present in the database or provided when the data was inserted. This can be done efficiently due to several previously described constraints:

- Data for each interval is contiguous in the bulkdata store.

```
function ExtractData(S_R, E_R, I)
    result ← [ ]                                          ▷ Initialize empty result list
    for all intervals i in I do                           ▷ For each interval
        (S, E, ρ_S, ρ_E) ← i                              ▷ Get interval parameters
        if S ≥ S_R then          ▷ If this interval starts within the requested range,
            ρ_{S_R} ← ρ_S                                 ▷ Use its start row
        else
            ρ_{S_R} ← LocateTime(S_R, ρ_S, ρ_E)           ▷ Locate start row
        if E ≤ S_E then          ▷ If this interval ends within the requested range,
            ρ_{S_E} ← ρ_E                                 ▷ Use its end row
        else
            ρ_{S_E} ← LocateTime(S_E, ρ_S, ρ_E)           ▷ Locate end row
        n ← ρ_{S_R}
        while n < ρ_{S_E} do                  ▷ For each row in matched range,
            result ← result + bulkdata[n]                 ▷ Append row to results
            n ← n + 1
    return result
```

Algorithm 2-1: Data extraction from an arbitrary time range $[S_R \rightarrow E_R\rangle$, given the set I of all stream intervals that intersect this range.

- Data timestamps are monotonically increasing, so the bulkdata storage for each interval is sorted by timestamp.
- Interval lookups can be done efficiently using the interval tree.

To extract the data, the NilmDB database layer first uses the interval tree to locate the set of all intervals I that intersect $[S_R \rightarrow E_R\rangle$. These three parameters are passed to ExtractData, shown in Algorithm 2-1. ExtractData then internally uses LocateTime, shown in Algorithm 2-2, to efficiently locate rows corresponding to specific timestamps in the bulkdata as necessary. These rows are returned through the serializer and server layers to the requesting client.

2.3.4 Bulkdata Storage

The majority of NilmDB data is handled by the "bulkdata" storage system. Bulkdata is an addressable row store, meaning that each sample of NILM data is stored under, and can be retrieved by, a unique row number. It provides three fundamental operations:

```
function LocateTime(t, ρ_S, ρ_E)
    ρ_low ← ρ_S                                  ▷ Initial search range is all of the rows
    ρ_high ← ρ_E
    while ρ_low < ρ_high do        ▷ Repeat until range has been narrowed to one row
        ρ_mid ← ⌊(ρ_low + ρ_high)/2⌋                                  ▷ Find midpoint
        if bulkdata[ρ_mid].timestamp < t then
            ρ_low ← ρ_mid + 1                          ▷ Narrow search to right half
        else
            ρ_high ← ρ_mid                             ▷ Narrow search to left half
    return ρ_low
```

Algorithm 2-2: Binary search to locate the first bulkdata row in the range $[\rho_S, \rho_E\rangle$ with a timestamp greater than t.

- Extract data from a specified row number.
- Insert new data, and return the row numbers corresponding to the starting and ending rows of that new data.
- Remove data corresponding to a range of row numbers.

The rows of data are stored as raw binary data on disk. The format of each row is derived from the stream layout, and matches the binary format described in the documentation of `/stream/insert` on page 109. Since streams contain only homogeneous data, this means that each row takes up the same number of bytes in the data file, denoted $B_{\texttt{row\_size}}$.

The structure of the bulkdata storage on disk is shown in Figure 2-9. Stream paths, such as `/stream/one`, are used as a directories in the filesystem. The data itself is stored in numbered files inside these directories. The number of rows in each data file, $N_{\texttt{rows\_per\_file}}$, is determined automatically at stream creation time, based on stream layout, so that the data file size is approximately 128 MiB. Depending on the underlying operating system, there may be a limit to how many files can efficiently be stored in a single directory. To avoid this issue, the data files are further grouped inside numbered directories. The group size is typically $N_{\texttt{files\_per\_dir}} = 32768$.

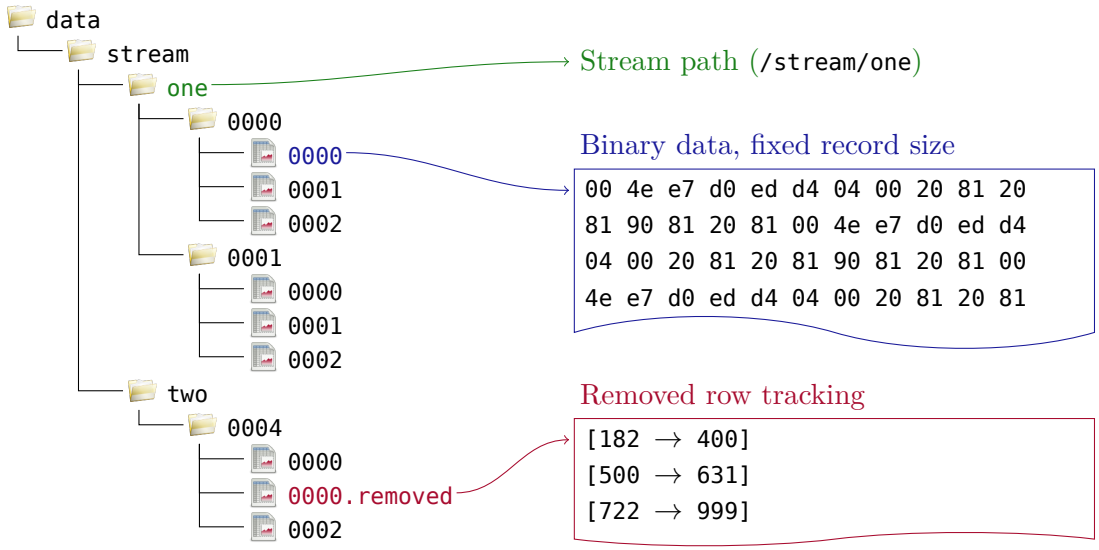


Figure 2-9: Filesystem structure for the Bulkdata storage engine. Stream paths are mirrored as a directory tree. Within each path, numbered directories group the data files into smaller blocks. Each data file contains binary data with a fixed record size. Files may be entirely removed, or partial removals can be marked in an additional tracking file.

2.3.4.1 Bulkdata Row Extraction

Extraction of data from a particular row or range of rows is straightforward. Due to the interval management, the higher levels of NilmDB will only request data that is known to be present in the bulkdata storage. Because each row takes up a fixed amount of space in the binary storage, the location of row n in stream `path` is fixed. It can be calculated as:

$$\texttt{group\_num} = \left\lfloor \lfloor n / N_{\texttt{rows\_per\_file}} \rfloor / N_{\texttt{files\_per\_dir}} \right\rfloor \tag{2.3}$$

$$\texttt{file\_num} = \mathrm{mod}\left(\lfloor n / N_{\texttt{rows\_per\_file}} \rfloor, N_{\texttt{files\_per\_dir}} \right) \tag{2.4}$$

$$\texttt{file\_offset} = \mathrm{mod}(n, N_{\texttt{rows\_per\_file}}) \cdot B_{\texttt{row\_size}} \tag{2.5}$$

The binary data for the row is then read out as the $B_{\texttt{row\_size}}$ bytes at offset `<file_offset>` in the file `path/<group_num>/<file_num>`. If the client request was for binary data, these bytes are returned directly; otherwise, the data is converted to ASCII text format first.

2.3.4.2 Bulkdata Row Insertion

The bulkdata storage is append-only; that is, all newly inserted data is appended to the last existing file, and the corresponding row numbers for the new data will be greater than any other row numbers in the stream. The system tracks the maximum row number $N_{\texttt{max}}$ ever used for a particular stream. When inserting m rows of data, the new data is written to file offsets corresponding to row $(N_{\texttt{max}}+1)$ through $(N_{\texttt{max}}+m)$, with file offsets calculated as they were for data extraction, and $N_{\texttt{max}}$ is updated accordingly.

This approach greatly simplifies the management of the bulkdata and avoids fragmentation of inserted intervals. It has the apparent drawback that, if new data is continuously inserted while old data is removed, the row numbers and $N_{\texttt{max}}$ grow without bound. In practice, this is not a concern, as the row numbers are internally stored as 64-bit integers, with a maximum of $2^{64}-1$. If data is continuously inserted at 100 KHz, the row numbers will increase for 6×10^{6} years before overflowing. By this time, the group directory names will also have grown from the default of 4 characters to a maximum of only 9.

2.3.4.3 Bulkdata Row Removal

At the bulkdata storage level, the goal of removal is to reclaim disk space. Other higher layers of NilmDB already handle removal of data in specific time ranges by truncating, splitting, or deleting intervals, adjusting the bulkdata row number pointers as necessary. These operations leave rows in the bulkdata storage that will never be accessed again, since row numbers are always increasing without reuse. The bulkdata removal process tracks these unused rows and, to the extent possible, marks them as free so that the operating system can reclaim the disk space.

The removal process is as follows. Every data file, such as `0000`, can have an associated row tracking file, `0000.removed`. This file contains a serialized representation of a list, created in the Python "pickle" format. Each entry in this list is a pair of row numbers `[start, stop]`, indicating a range of "removed" rows that are no longer

referenced by any intervals and will no longer be accessed. As more rows are removed, more entries are added to this file. Finally, when every row in a particular data file has been marked as removed, both the data file and its tracking file are completely deleted from disk, freeing the space previously used.

Note that, until all of the rows in a particular data file have been marked as removed, disk space may not be reclaimed. For example, deleting every other row of data will never free disk space, because the data file will never be completely deleted. In general usage, however, when larger blocks of data are deleted, or old data is deleted as new data is added, the "wasted" space should remain small. The expected wasted storage is on the order of the data file size (128 MiB). NilmDB also attempts to mitigate this wastage by informing the underlying operating system when specific ranges of files will no longer be used. Modern filesystems, like ext4 in Linux, use this to "punch holes" in the data files, which can free disk space even before the data file is deleted.

2.4 NILM Manager

NILM Manager, developed by John Donnal [33], is a unified and centralized management infrastructure for the distributed NilmDB system. It connects to remote NilmDB systems through a secure virtual private network (VPN), providing a simple, user-friendly and web-based interface to any authenticated user with an Internet connection. This interface includes a wide variety of tools, including configuration of servers and streams, interactive and real-time data visualizations, and the execution and management of data filters and processing tasks; most importantly, it facilitates performing these actions with a minimum of required network bandwidth to the remote NilmDB.

Figure 2-10 shows a screenshot of the NILM Manager homepage and some of its available tools and applications. This web interface, developed with HTML5 and Javascript, is supported by a backend "Ruby on Rails" server application that acts as a connection and data broker to individual NilmDB systems, and maintains

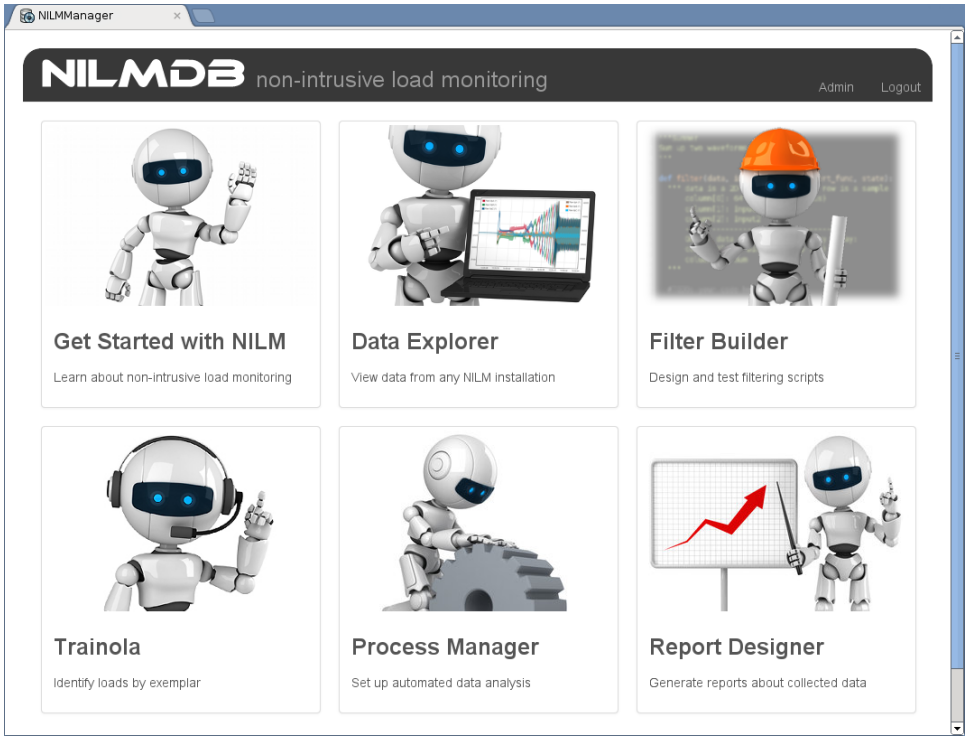


Figure 2-10: NILM Manager overview. A variety of tools and interfaces are provided for working with and manipulating a network of NilmDB systems and data.

overall system configuration and state. The manager also provides a comprehensive administrative interface, demonstrated in Figure 2-11, for building and maintaining this configuration.

While the NILM Manager is a critical part of a worldwide distributed NilmDB system, a complete description of its use and implementation is outside the scope of this thesis. Instead, the following sections focus on three specific components of the NilmDB framework that were developed to support NILM Manager tools. The first, `nilm-decimate`, is a filter that reduces stream data to a form suitable for efficient data visualizations of the "Data Explorer" tool. The second, NilmRun, supports the "Process Manager" tool by providing the ability to execute and manage filters and other processes on a remote NilmDB machine. The third, `nilm-trainola`, is a load identification filter that is used by the "Trainola" tool to disaggregate and identify individual transient events within a stream.

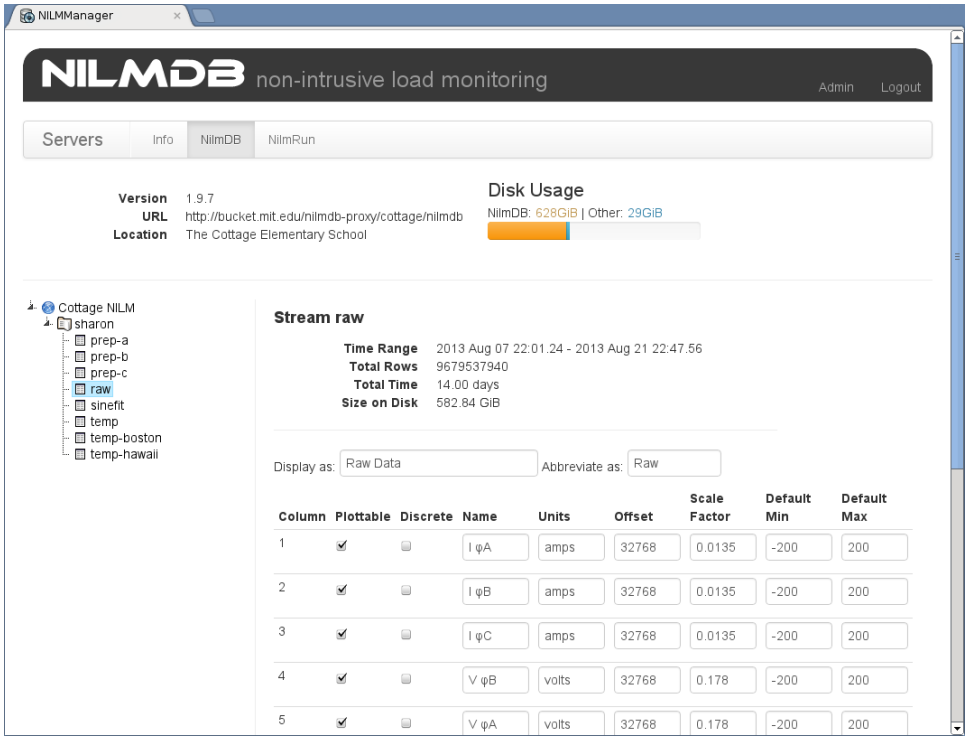


Figure 2-11: NILM Manager administrative interface, which supports configuration and monitoring of NilmDB systems and their streams.

2.4.1 Data Visualization and Decimation

The manager provides a powerful data visualization and navigation interface, shown in Figure 2-12. This interface forms a central component of the manager, and is used both as a standalone tool for exploring NilmDB streams, and as an embedded component for controlling and visualizing the output of other tools. The features of the plot engine include:

- Live, draggable, zoomable plots with "Google Maps-like" navigation.
- Dual y axes with independently adjustable scaling and positioning.
- Simultaneous plotting of data with compatible units.
- Overview window to facilitate navigation of large data streams.
- Save and load support to recall specific streams and views.
- Continuous and discrete event plotting styles.

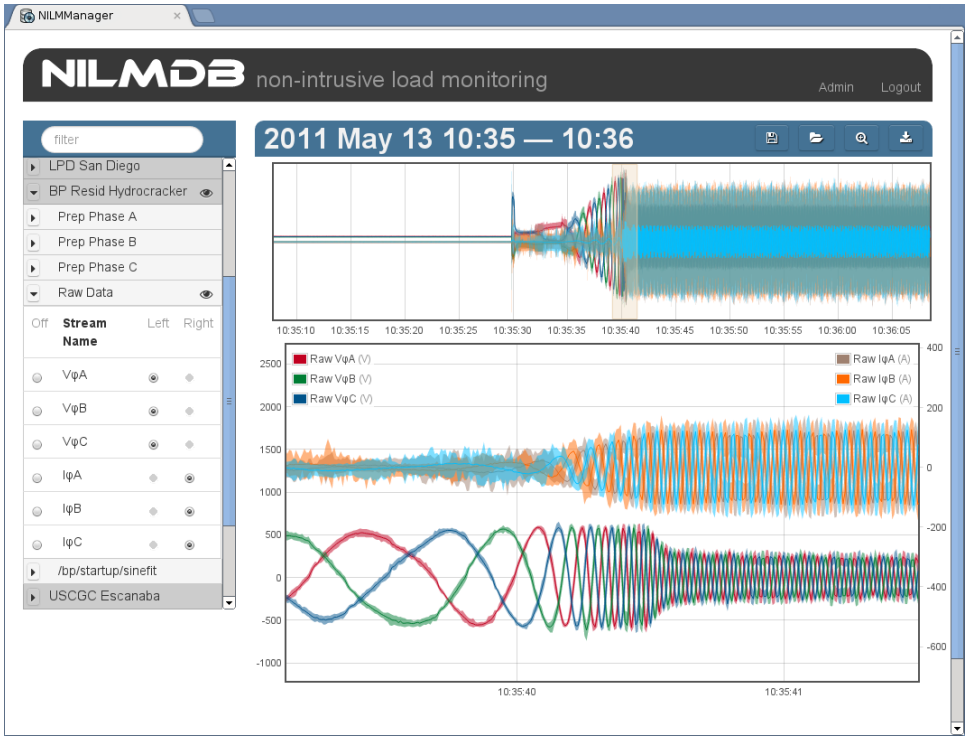


Figure 2-12: NILM Manager data explorer and visualization. Stream data can be navigated and zoomed in real-time, with minimal network data transfer.

- Antialiasing of high frequency data through shading of signal envelopes.
- Dynamic scaling and resizing based on stream data and client window geometry.
- Support for discontinuous data and gaps in streams.

Crucially, the plotting engine achieves these features while transferring only a minimal amount of data from the remote NilmDB server. Typically, the number of data points retrieved to display a particular window of data is on the order of 1,000 per plotted stream, regardless of zoom level.

The plotting engine achieves these low data transfer rates in two ways. First, it makes heavy use of the stream and interval support in the NilmDB HTTP API, particularly when extracting data. The plot uses operations such as the "count" option of `/stream/extract`, detailed in Section 3.2.1.14, to verify the amount of data that will be returned for a particular time range. Discontinuous data and gaps are supported through the use of the "markup" option, which avoids the need to send multiple requests for each interval. The plot engine can simply request all data

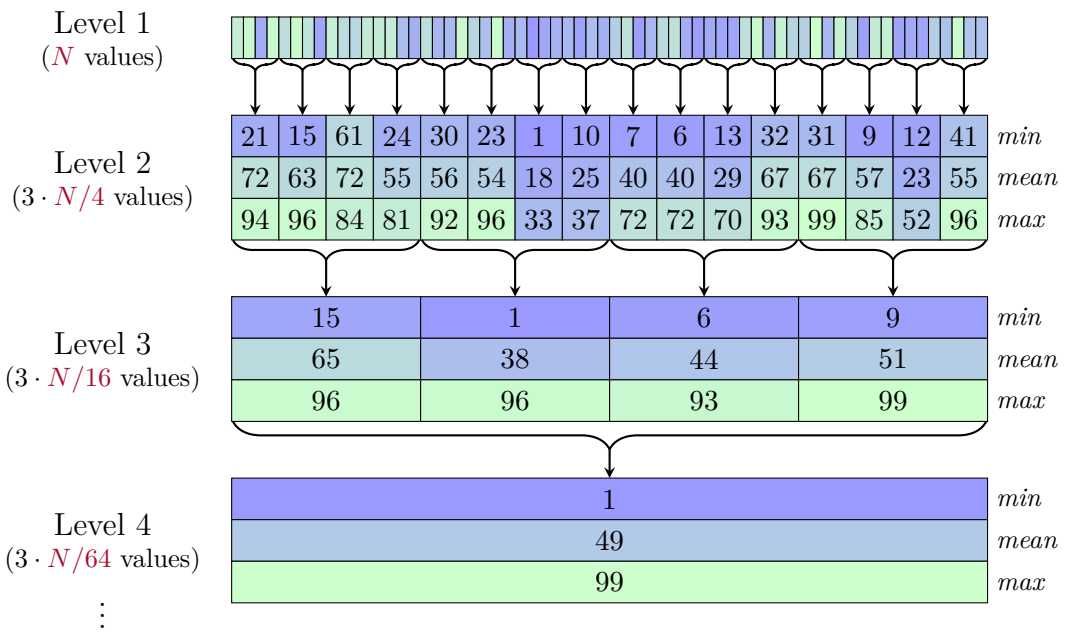


Figure 2-13: Decimation of stream data. Each decimation level tracks the minimum, mean, and maximum of a block of values from the previous level. NILM Manager chooses the appropriate level when requesting data to plot. Regardless of the number of decimations, the total storage requirement for N original values is only $2N$.

from multiple streams over arbitrary regions of time, corresponding to the currently displayed x-axis, and the server manages the details of finding and returning only that data which is needed.

The second feature that enables efficient plotting is decimation. Here, decimation is a process by which ancillary streams of filtered, downsampled data are precomputed and stored on the server, similar to the computer graphics technique of "MIP mapping" [40]. An example of the decimation process is shown in Figure 2-13.

Decimated streams store the mean value for each column of an input stream, including the timestamp, calculated over small successive blocks of γ rows. Typically, $\gamma = 4$. Thus, for an input stream with N rows, the first decimation contains $N/4$ rows. The process can be repeated in multiple "levels", with each level having correspondingly fewer rows, until just one row remains, containing the average of all the data in the stream. In addition to the mean, decimated streams also store the minimum and maximum values of each successive block. For repeated decimations, these are calculated over the previously computed extrema.

When a requesting data for a plot, NILM Manager queries the server for the total number of data rows in the desired interval. Based on the response, it automatically determines and requests data from the decimation level that contains the optimal number of points for display. The means are plotted as a line, and the minima and maxima are used to plot signal envelopes in a lighter shade. This helps maintain a visual indication of the data range of the original stream, similar to the display of a digital oscilloscope. The averaging operation also provides a simple low-pass filter, removing aliasing effects from the plot.

The additional storage requirements for the decimated streams are modest. Consider a stream with N rows and one column per row that is decimated by a factor of 4, as shown in Figure 2-13. Decimated streams store three times as many columns (minimum, mean, and maximum) as the original stream, but each decimation level contains one-fourth as many rows. The total number of stored values for the original data plus L decimation levels is given by the geometric series:

$$N_{\text{total}} = N + 3N \cdot \sum_{k=1}^{L} \left(\frac{1}{4}\right)^k \tag{2.6}$$

$$= 3N \cdot \sum_{k=0}^{L} \left(\frac{1}{4}\right)^k - 2N \tag{2.7}$$

Taking the limit of this as $L \to \infty$ gives:

$$\lim_{L\to\infty} N_{\text{total}} = 3N \cdot \left(\frac{1}{1-(1/4)}\right) - 2N \tag{2.8}$$

$$= 2N \tag{2.9}$$

Therefore, storing *every* decimation level of a stream in NilmDB will at most only double the storage requirements of the original data, when decimating by a factor of 4. This overhead is low enough that it is almost always outweighed by the resulting bandwidth reduction and visual quality of the plots.

Decimation is performed by the `nilm-decimate` filter and the associated `nilm-decimate-auto` program, which automatically computes all required levels of deci-

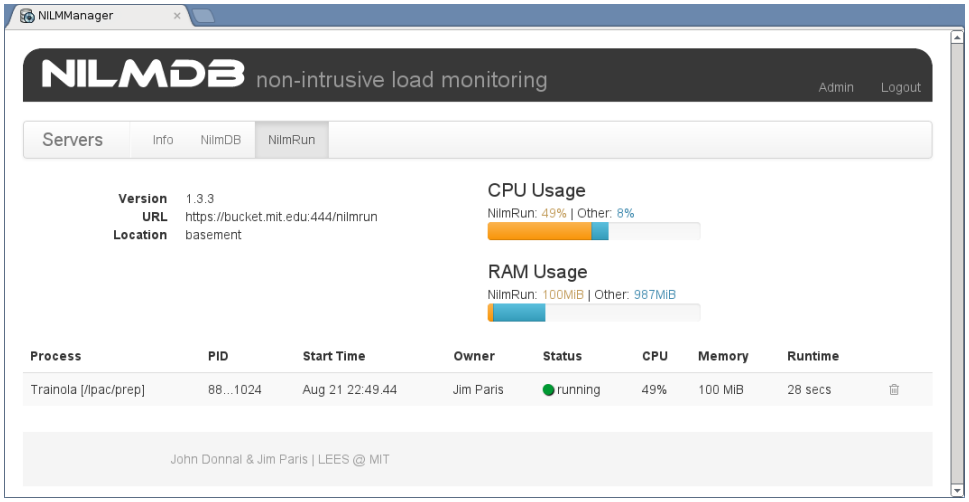


Figure 2-14: NILM Manager remote process management. Filters and other scripts are executed on remote NilmDB machines via the NilmRun server.

mation for multiple streams. The usage of these programs is documented in Sections 3.4.2.4 and 3.4.2.5.

2.4.2 Remote Process Management and NilmRun

In order to fully support the distributed computation model of NilmDB, the NILM Manager provides the ability to control the execution of processes on NilmDB hosts. This allows a user to conserve bandwidth and increase computation throughput by transmitting short filters and other programs to a remote machine, executing them there, and retrieving the status and results. These remote processes can interact directly with the remote NilmDB, extracting data and inserting results as streams.

Figure 2-14 shows the process management control panel within the NILM Manager administrative interface. Remote process execution is facilitated by a NilmDB component called NilmRun. NilmRun is an HTTP-based server application that mirrors the architecture of the NilmDB server, providing a client API interface with a series of commands for starting, stopping, and querying the status of processes. The server runs alongside NilmDB; while it can run independently, it is typically served up by the same Apache instance, and can share the networking and access control setup used by NILM Manager for the usual NilmDB connection.

NilmRun supports executing existing programs on the remote machine, as well as arbitrary blocks of Python code. The former is primarily focused on running existing tools within the NilmDB framework, such as the spectral envelope preprocessor (Chapter 4), while the latter is geared towards new custom filters and reports developed within NILM Manager. In both cases, interfaces are provided for retrieving the text output and exit codes from the program, which NILM Manager uses to provide progress information and indicate success.

Besides direct process control, the NilmRun server also provides system-wide information such as CPU, I/O, and memory usage statistics. This can be used to help determine how much processing capability is available or remaining on a particular NilmDB system. For large or complex NILM deployments, multiple computers with NilmRun could be installed on-site. Future versions of NILM Manager could then use these statistics to automatically distribute jobs among the available machines.

The NilmRun API, and a suite of command-line tools to list and control processes running on the remote server, are further documented in Sections 3.3.1 and 3.3.2.

2.4.3 "Trainola" Transient Event Identification

One of the defining aspects of non-intrusive load monitoring is sensor reduction through the acquisition of aggregate power measurements from a collection of loads. Individual load identification and diagnostics requires subsequent disaggregation of these loads. Generally, this relies on the existence of some unique metric or feature of individual systems that distinguishes between the loads of interest. Typical metrics include steady-state power levels and transient event shape, amplitude, and sequencing. In particular, event identification based on exemplar matching has been demonstrated as a particularly useful technique for identification and diagnostic monitoring [2, 4, 11, 13]. To support this, NilmDB and NILM Manager provide the "Trainola" transient event identification tool.

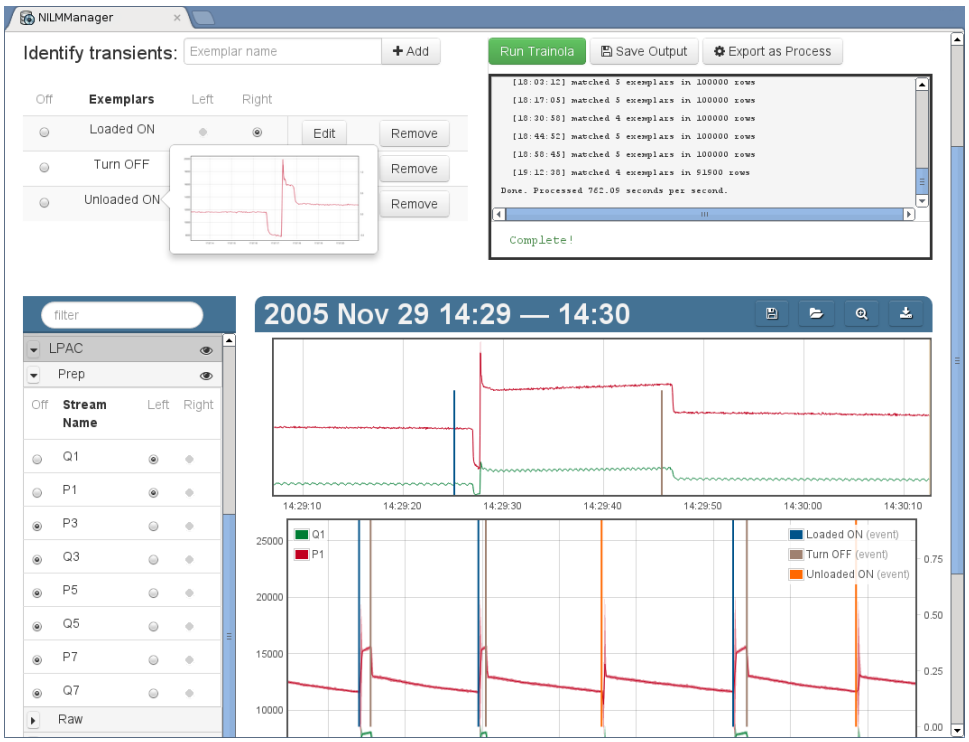


Figure 2-15: "Trainola" exemplar-based event identification. The user graphically identifies examples of transient events in an input stream, which can then be automatically located and marked in an output stream. Matching events are plotted as vertical lines, overlaid on the input data.

2.4.3.1 User Interface

Trainola is exposed by NILM Manager as an interactive workspace, shown in Figure 2-15. The lower half of the window mirrors the Data Explorer interface, where the user can freely select, plot, and navigate NilmDB streams. At the top, the user can create and name exemplars based on the data currently in view. Typically, these exemplars correspond to particular "turn-on" and "turn-off" events, and extend to include a few seconds of steady-state behavior before and after each transient event. For streams with multiple columns, such as spectral envelope preprocessor harmonics, the exemplars consist of whichever columns are visible when the exemplar is saved.

To run the automatic identification process, the user visually navigates to the target data stream and zooms out to the time-frame over which events should be identified. The exemplars and target data do not have to come from the same stream, but the same named columns must be present in both. Then, the "Run Trainola"

button starts a `nilm-trainola` process on the remote machine, which will identify events and continue to run until completion or cancellation, even if the web browser is closed. During and after the identification, matched events can be viewed as vertical lines overlaid on the plot, by selectively enabling each exemplar. The events are also stored in a dedicated NilmDB output stream, and can be accessed by other filters and tools.

For continuous load identification on new data, `nilm-trainola` can also be executed from the command line, given that the correct exemplars have been identified. Data formats and usage of the command line tool are detailed in Section 3.4.2.11.

2.4.3.2 Matching Algorithm

The `nilm-trainola` tool matches the shape of exemplars to the input data using the following algorithm. Consider two sampled waveforms of equal size N, for example, an observation $f[n]$ and an exemplar $g[n]$. After removing dc offset, a measure of similarity between two waveforms is the Euclidean distance, defined as:

$$D = \sum_{n \in N} \left(f[n] - g[n]\right)^2 \tag{2.10}$$

This expression can be expanded to:

$$D = \sum \left((f[n])^2 - 2f[n]g[n] + (g[n])^2\right) \tag{2.11}$$

$$= \sum f[n]^2 + \sum g[n]^2 - 2\sum f[n]g[n] \tag{2.12}$$

which is more conveniently expressed in terms of the dot product:

$$D = (f \cdot f) + (g \cdot g) - 2(f \cdot g) \tag{2.13}$$

$$= |f|^2 + |g|^2 - 2(f \cdot g) \tag{2.14}$$

If the waveforms match, the Euclidean distance between them would be $D = 0$, and so the equation reduces to:

$$0 = |f|^2 + |g|^2 - 2(f \cdot g) \tag{2.15}$$

$$f \cdot g = \frac{|f|^2 + |g|^2}{2} \tag{2.16}$$

Furthermore, if the amplitudes match, $|f| = |g|$, giving:

$$f \cdot g = \frac{2|g|^2}{2} \tag{2.17}$$

$$\frac{f \cdot g}{|g|^2} = 1 \tag{2.18}$$

Thus, (2.18) holds when the two waveforms match in amplitude and shape. Non-matching waveforms may also satisfy this condition in degenerate cases, but in general, $M = (f \cdot g)/|g|^2$ has been found to be a useful figure of merit to use when judging power signature similarity [13]. As M approaches 1.0, it indicates confidence that f and g match, both in shape and amplitude.

When the full waveform f contains more points than the exemplar g, the calculation can be performed over sliding windows of the input data, determining $M(t)$ at each window offset. As g "slides" over a feature in the f that matches in shape and amplitude, $M(t)$ will approach a local maximum of 1.0. Figures 2-16 and 2-17 demonstrate this metric for both a matching and non-matching exemplar case. In `nilm-trainola`, a peak-finding algorithm is applied to locate local maxima, and values that fall within a small detection window around 1.0 are marked as matched events. To reduce false positives, peaks that are close to either minima or maxima with larger absolute magnitudes are ignored.

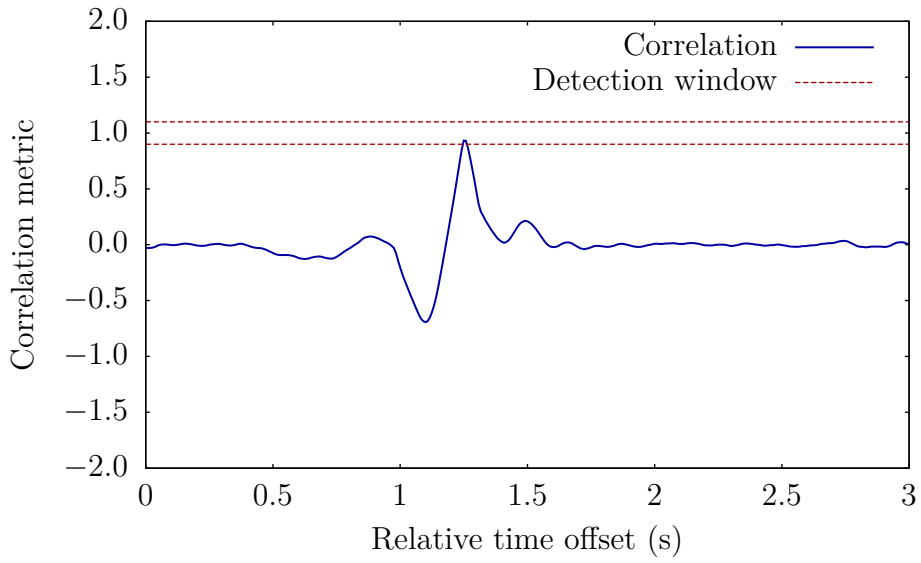


Figure 2-16: Correlation metric for a matching exemplar. A peak that falls within a detection window around 1.0 indicates that the exemplar matches at that time.

2.4.3.3 Cross-Correlation

$M(t)$ can be computed more efficiently than (2.18) by using the cross-correlation [41]. For two waveforms x and y, the cross-correlation is defined as:

$$(x \star y)[t] = \sum_{k=-\infty}^{\infty} x^*[k]\ y[t+k] \tag{2.19}$$

where x^* denotes the complex conjugate of x. For real numbers, this represents the same basic operation as the dot product $(x \cdot y)$. Let x be the input waveform, and let y be the exemplar with the dc offset removed. Then, the correlation metric $M(t)$ can be written as:

$$M(t) = \frac{x \star y}{|y|^2} \tag{2.20}$$

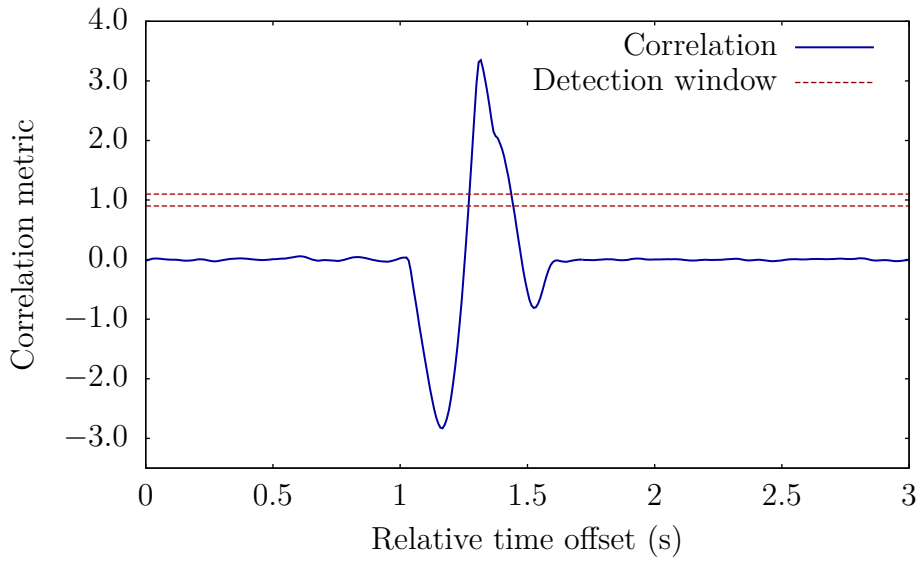


Figure 2-17: Correlation metric for a non-matching exemplar. The correlation metric is near 1.0 twice, but is not considered a match because this does not occur at a local maximum.

Cross-correlations can be computed quickly with the discrete Fourier transform $\mathcal{F}$ using the relation:

$$\mathcal{F}\{x \star y\} = \mathcal{F}^*\{x\}\mathcal{F}\{y\} \tag{2.21}$$

or, solving for the convolution:

$$x \star y = \mathcal{F}^{-1}\left(\mathcal{F}^*\{x\}\mathcal{F}\{y\}\right) \tag{2.22}$$

This is used by `nilm-trainola` to efficiently match over large input streams.

Note that in (2.10), dc offset was removed from both the input and the exemplar; here, the dc offset is only removed from the exemplar. However, (2.21) demonstrates that this is equivalent: the product of the two DFTs will have a dc component of zero as long as either x or y has zero dc component.

2.4.3.4 Matching Multiple Columns

Trainola supports exemplar matching on multiple columns of an input stream. For a stream like the output of the spectral envelope preprocessor, columns represent specific in-phase and quadrature harmonics, such as P_1, Q_1, and P_3. Within NILM Manager, the user includes these additional columns in exemplars by simply having them visible when the exemplar is created. Including multiple harmonics when classifying loads in an aggregate measurement can significantly improve the accuracy of identification, as different types of loads may have similar signatures in some harmonics but vary in others.

The matching algorithm of Section 2.4.3.2 is extended to N columns by considering each one independently, and assigning a higher importance to the columns with a larger magnitude in the exemplar. Specifically, let A_n be the magnitude of each of the exemplar columns, and number the columns in descending order such that A_1 is the largest and A_N is the smallest. The correlation metric $M_n(t)$ is calculated for each column, and peak-finding is performed on $M_1(t)$ to locate the maxima. For every candidate peak t_P, the following must hold true for a successful match:

$$\frac{0.9A_n}{A_1} \leq M_n(t_P) \leq 2 - \frac{0.9A_n}{A_1} \qquad \forall n \in N \tag{2.23}$$

This formula represents a linear "widening" of the detection window in Figure 2-16 as the relative column magnitude decreases. The largest column must have correlation values that are within 1.0 ± 0.1, while a column with half this magnitude must have a correlation of 1.0 ± 0.45. This allows for more variation in the columns that are smaller and are more likely to have been affected by noise, or vary between transients. When all values $M_n(t_P)$ satisfy this condition, `nilm-trainola` marks a successful exemplar match in the output stream.

Chapter 3

NilmDB Reference Guide

This chapter provides a reference guide to setting up, interfacing, and managing a NilmDB system. The layered architecture of NilmDB, described in Section 2.3, provides a variety of means in which a user may interface with NilmDB, depending on the target application. Each of these layers is defined by a particular application programming interface (API), which the external software must follow. For example, the NILM Manager in Section 2.4 utilizes the lowest level HTTP API to fetch stream data and metadata for use in data visualization, while interactive or scripted management of the database will likely use the command line reference instead. Because each layer serves different purposes, they are all considered to be "supported" interfaces for NilmDB.

Section 3.1 discusses the process of building and installing the NilmDB server, the NilmRun server, and the NilmTools package of filters and tools. For NilmDB and NilmRun, Sections 3.1.1 and 3.1.2 additionally describe how to make the HTTP layer accessible through an Apache or other Web Server Gateway Interface (WSGI) compliant web server.

Sections 3.2.1 to 3.2.3 describe the HTTP API, Python API, and command-line interfaces to the main NilmDB database server. Section 3.3.1 describes the HTTP interface to the NilmRun process management server. Finally, Sections 3.4.1 and 3.4.2 describe the Python API and the command line interfaces for the NilmTools package of filters and tools.

3.1 Installation and Configuration

This section describes in detail how to download, build, configure, and install the NilmDB, NilmRun, and NilmTools packages. For an easier automated installation of a basic NilmDB system, based on the "NilmDBuntu" images, see Appendix A.

3.1.1 NilmDB

The NilmDB software is straightforward to install on a typical GNU/Linux software distribution such as Debian or Ubuntu. For a "bare-metal" install, begin by installing a recent version of one of these operating systems; the server edition of Ubuntu 12.10 is suggested. Then, specific instructions for each aspect of setting up NilmDB follow.

3.1.1.1 Building and Installation

The main NilmDB server has a number of external software dependencies. Table 3-1 lists all these dependencies, as well as the recommended or required version for each package. On a sufficiently up-to-date system, such as Ubuntu 12.10 or newer[1], the following commands will install all required packages:

```
# Install runtime and build environments
sudo apt-get install python2.7 python2.7-dev python-setuptools python-pip
sudo apt-get install cython git build-essential

# Base NilmDB dependencies
sudo apt-get install python-cherrypy3 python-decorator python-simplejson
sudo apt-get install python-requests python-dateutil python-tz
sudo apt-get install python-progressbar python-psutil

# Other dependencies for filters, automated tests, and web server setup
sudo apt-get install python-numpy
sudo apt-get install python-nose python-coverage
sudo apt-get install apache2 libapache2-mod-wsgi
```

[1]The current long-term support release of Ubuntu (12.04 LTS) ships a version of `cython` that is too old. Such systems can still be supported by creating a `.tar.gz` package of NilmDB on a more recent system using `make dist` in the source directory, then using that package on a target system in lieu of a git clone.

Package Name[1]	**Version**	**Description**
`python-nose`[2]	$\geq$ 1.1.2	Test framework
`python-coverage`[2]	$\geq$ 3.4	Test coverage reports
`python-numpy`[3]	$\geq$ 1.6	Numerical computing package
`python-setuptools`	$\geq$ 0.6	Package setup manager (distribute)
`python-pip`	$\geq$ 1.1	Python package manager
`python-decorator`	$\geq$ 3.4	Python decorator module
`python-cherrypy3`	$\geq$ 3.2	Web application server
`python-simplejson`	$\geq$ 2.5	Python JSON module
`python-dateutil`	$\geq$ 1.5	Python date manipulation module
`python-tz`	$\geq$ 2012c	Python timezone module
`python-psutil`	$\geq$ 0.3.0	Python process management module
`python-requests`	$\geq$ 1.1.0, $<$ 2	HTTP request library
`python-progressbar`[4]	$\geq$ 2.2	Python progress bar module
`python2.7`[5]	$\geq$ 2.7, $<$ 3	Python 2.x runtime
`python2.7-dev`[5]	$\geq$ 2.7, $<$ 3	Python headers, C compiler
`cython`[6]	$\geq$ 0.16	C extensions for Python
`git`[6]	$\geq$ 1.7	Git distributed version control
`apache2`[7]	$\geq$ 2.2	Apache HTTP server
`libapache2-mod-wsgi`[7]	$\geq$ 3.3	Apache WSGI module

[1] These names match the corresponding Debian/Ubuntu packages.
[2] Only required when running tests.
[3] Optional, but required by tests and other tools.
[4] Used by the `nilmdb-fsck` tool only.
[5] Python 3.x support would require some minor code updates throughout.
[6] Not needed if building from a `.tar.gz` package.
[7] Other WSGI compliant servers can be used.

Table 3-1: Software requirements for building and running a NilmDB server.

Once dependencies are installed, the source code for NilmDB can be retrieved via git. The following commands will clone the repository and check out a specific tagged revision:[2]

```
# Clone the repository into user's home directory
cd
git clone https://git.jim.sh/jim/lees/nilmdb.git
cd nilmdb

# Check out a specific revision (optional; omit for latest code)
git checkout nilmdb-1.9.7
```

The NilmDB software can then be installed system-wide with:

```
# Install
sudo make install
```

At this point, the programs `nilmtool`, `nilmdb-server`, and `nilmdb-fsck` will have been installed to `/usr/local/bin` and should be executable at the command line:

```
# Verify that installed programs are executable
nilmtool --version
```

After installation, running the built-in test suite is recommended. This will ensure that dependencies were installed correctly, and will verify the correctness of NilmDB on the filesystem and system architecture:

```
# Run tests
sudo make test
```

A report on code coverage and test results will be printed to the terminal. Successful tests will end with output similar to:

[2] If requested by the server, use username "`nilm`" and password "`nilm`".

```
TOTAL                        2323      0   100%
----------------------------------------------------------------------
Ran 68 tests in 31.569s

OK (SKIP=1)
```

3.1.1.2 Testing With the Built-in HTTP Server

For testing, or installations where running a full-featured web server is not desirable, NilmDB provides an embedded HTTP server using the CherryPy library. This mode of operation may be useful for testing, since the server can be manually started from within a terminal window using the `nilmdb-server` program. More details on invoking the standalone server are provided in Section 3.2.3.13. For simple testing, first choose a local filesystem directory that will hold the NilmDB database contents. Here, the directory `db` under the current user's home directory will be used. The server can be started with:

```
# Run NilmDB server
cd
nilmdb-server --database db
```

By default, the standalone server will listen on HTTP port 12380. The `nilmtool` client, detailed in Section 3.2.3.1, can then be executed in another terminal to verify connectivity and interact with NilmDB:

```
# Test connections to NilmDB server
nilmtool --url http://localhost:12380/ info
```

The expected output of this command follows the format:

```
Client version: 1.9.7
Server version: 1.9.7
Server URL: http://localhost:12380/
Server database path: /home/user/db
Server disk space used by NilmDB: 46 kiB
```

```
Server disk space used by other: 516.76 GiB
Server disk space reserved: 6.86 GiB
Server disk space free: 153.21 GiB
```

The embedded server is relatively high-performance, and can be used directly for normal NilmDB operations. However, running through a web server such as Apache greatly increases flexibility, and ensures that NilmDB is started automatically at system startup. Setting this up is described in the following section.

3.1.1.3 Configuring Apache WSGI Interface for NilmDB

NilmDB is a Web Server Gateway Interface (WSGI) compliant application, which allows it to be served by any web server that supports WSGI. Here, instructions are given for configuring the NilmDB application to run under Apache, using the `mod_wsgi` module on a Debian or Ubuntu system. Configuration for other systems will be similar.

First, ensure that the Apache installation works and is accessible at `http://host/`, where *`host`* is the hostname or IP address of the system. The NilmDB server will be set up at address `http://host/nilmdb`. It will run as as user `nilmdb`, group `nilmdb`, and the database will be stored in the local filesystem directory `/home/nilmdb/db`.

If user `nilmdb` does not already exist, it can be created with:

```
# Create user, group, and home directory
sudo adduser nilmdb

# Lock the password (if user nilmdb is not intended to login directly)
sudo passwd -l nilmdb
```

A wrapper script `/home/nilmdb/nilmdb.wsgi` should be created as a plain text file containing:

```
import nilmdb.server
application = nilmdb.server.wsgi_application("/home/nilmdb/db","/nilmdb")
```

The first string in the wrapper script is the local filesystem directory, and the second parameter is the path portion of the final URL. The script should be owned by user `nilmdb`:

```
# Set file owner and group
sudo chown nilmdb:nilmdb /home/nilmdb/nilmdb.wsgi
```

Then, Apache can be configured to point to the wrapper script. The default Apache configuration resides in `/etc/apache2/sites-enabled/000-default`, and the following lines should be added within the `<VirtualHost>` directive:

```
<VirtualHost *:80>
    # Add these 6 lines
    WSGIScriptAlias /nilmdb /home/nilmdb/nilmdb.wsgi
    WSGIDaemonProcess nilmdb-procgroup threads=32 user=nilmdb group=nilmdb
    <Location /nilmdb>
        WSGIProcessGroup nilmdb-procgroup
        WSGIApplicationGroup nilmdb-appgroup
    </Location>

    # (other existing configuration here)
</VirtualHost>
```

The Apache server must be restarted for the changes to take effect:

```
# Restart Apache
sudo apache2ctl restart
```

More complex configuration, including SSL encryption, authentication, proxying, compression, or path-based access control with the `<Location>` option, is independent of NilmDB, and can be added using standard Apache configuration directives [42].

The NilmDB server should now be running under Apache. It can be tested by visiting `http://`*`host`*`/nilmdb/` in a web browser. Any errors will be reported in the web browser, or in the Apache log file `/var/log/apache2/error.log`. On success, the web browser will see a page containing:

Package Name[1]	Version	Description
`nilmdb`	$\geq 1.9.7$	NilmDB Framework
`python-nose`[2]	$\geq 1.1.2$	Test framework
`python-coverage`[2]	≥ 3.4	Test coverage reports
`python-setuptools`	≥ 0.6	Package setup manager (distribute)
`python-pip`	≥ 1.1	Python package manager
`python-cherrypy3`	≥ 3.2	Web application server
`python-simplejson`	≥ 2.5	Python JSON module
`python-psutil`	$\geq 0.3.0$	Python process management module
`python2.7`[3]	$\geq 2.7, < 3$	Python 2.x runtime
`git`[4]	≥ 1.7	Git distributed version control
`apache2`[5]	≥ 2.2	Apache HTTP server
`libapache2-mod-wsgi`[5]	≥ 3.3	Apache WSGI module

[1]These names match the corresponding Debian/Ubuntu packages.
[2]Only required when running tests.
[3]Python 3.x support would require some minor code updates throughout.
[4]Not needed if building from a `.tar.gz` package.
[5]Other WSGI compliant servers can be used.

Table 3-2: Software requirements for building and running a NilmRun server.

```
This is NilmDB version 1.9.7, running on host hostname.
```

At this point, the NilmDB installation is fully functional, and the server address `http://`*`host`*`/nilmdb/` can be used as the URL for `nilmtool` and related commands, or added to an existing instance of NILM Manager (Section 2.4) as a new server.

3.1.2 NilmRun

Installing and configuring the NilmRun server is similar to the installation of NilmDB described in Section 3.1.1. NilmRun depends on software components provided by NilmDB, and so NilmDB must be installed first. However, the servers themselves are independent, and it is not necessary to *run* NilmDB in order to run the NilmRun server.

3.1.2.1 Building and Installation

Table 3-2 lists the external software dependencies for NilmRun. Note that many dependencies are shared between NilmDB, and following the installation process in

Section 3.1.1.1 should install all of the required software for NilmRun. Once dependencies are installed, the source code for NilmDB can be retrieved via git. The following commands will clone the repository and check out a specific tagged revision:

```
# Clone the repository into user's home directory
cd
git clone https://git.jim.sh/jim/lees/nilmrun.git
cd nilmrun

# Check out a specific revision (optional; omit for latest code)
git checkout nilmrun-1.3.3
```

The NilmRun software can then be installed system-wide with:

```
# Install
sudo make install
```

At this point, the programs `nilmrun-server`, `nilmrun-ps`, `nilmrun-run`, and `nilmrun-kill` will have been installed to `/usr/local/bin` and should be executable at the command line.

After installation, running the built-in test suite is recommended. This will ensure that dependencies were installed correctly and that the NilmRun process manager works correctly:

```
# Run tests
sudo make test
```

A report on code coverage and test results will be printed to the terminal. Successful tests will end with output similar to:

```
TOTAL                     261      0   100%
----------------------------------------------------------------------
Ran 9 tests in 20.510s

OK (SKIP=1)
```

3.1.2.2 Testing With the Built-in HTTP Server

NilmRun provides an embedded HTTP server using the CherryPy library. This mode of operation may be useful for testing, since the server can be manually started from within a terminal window using the `nilmrun-server` program. Usage is as follows:

```
# Run NilmRun server
nilmrun-server
```

By default, the standalone server will listen on HTTP port 12381. Note that this server contains no built-in security, and the nature of NilmRun means that anybody that can connect to the server can execute software on it. Running the server through Apache, and utilizing its security features, is therefore strongly recommended.

3.1.2.3 Configuring Apache WSGI Interface for NilmRun

NilmRun is a Web Server Gateway Interface (WSGI) compliant application, which allows it to be served by any web server that supports WSGI. Here, instructions are given for configuring the NilmRun application to run under Apache using the `mod_wsgi` module on a Debian or Ubuntu system. The NilmRun server, and any programs executed through NilmRun, will be configured to run as the `nilmdb` user, which was set up in Section 3.1.1.3. The address of the server will be `http://`*`host`*`/nilmrun`.

First, a wrapper script `/home/nilmdb/nilmrun.wsgi` should be created as a plain text file containing:

```
import nilmrun.server
application = nilmdb.server.wsgi_application("/nilmrun")
```

The parameter in the wrapper is the path portion of the final URL. This wrapper script should be owned by user `nilmdb`:

```
# Set file owner and group
sudo chown nilmdb:nilmdb /home/nilmdb/nilmrun.wsgi
```

Then, Apache can be configured to point to the wrapper script. The default Apache configuration resides in `/etc/apache2/sites-enabled/000-default`, and the following lines should be added within the `<VirtualHost>` directive:

```
<VirtualHost *:80>
    # Add these 6 lines
    WSGIScriptAlias /nilmrun /home/nilmdb/nilmrun.wsgi
    WSGIDaemonProcess nilmrun-procgroup threads=32 user=nilmdb group=nilmdb
    <Location /nilmrun>
        WSGIProcessGroup nilmrun-procgroup
        WSGIApplicationGroup nilmrun-appgroup
    </Location>

    # (other existing configuration here)
</VirtualHost>
```

The Apache server must be restarted for the changes to take effect:

```
# Restart Apache
sudo apache2ctl restart
```

Note that, since NilmRun allows the execution of arbitrary code, it is *strongly* recommended that precautions be taken to limit who can connect to the server. For example, requiring SSL and adding HTTP Basic Authentication [43] will effectively limit access. If a VPN is used, IP-based restrictions can be applied in the `<Location>` directive [42].

The NilmRun server should now be running under Apache. It can be tested by visiting `http://host/nilmrun/` in a web browser. Any errors will be reported in the web browser, or in the Apache log file `/var/log/apache2/error.log`. On success, the web browser will see a page containing:

```
This is NilmRun version 1.3.3, running on host hostname.
```

The URL can then be added to NILM Manager (Section 2.4) to enable process management and filter execution on this server.

Package Name[1]	Version	Description
`nilmdb`	$\geq 1.9.7$	NilmDB Framework
`python-setuptools`	≥ 0.6	Package setup manager (distribute)
`python-pip`	≥ 1.1	Python package manager
`python-simplejson`	≥ 2.5	Python JSON module
`python2.7`[2]	$\geq 2.7, < 3$	Python 2.x runtime
`git`[3]	≥ 1.7	Git distributed version control
`python-numpy`	≥ 1.6	Numerical computing package
`python-scipy`	≥ 1.7	Scientific computing package
`python-daemon`	≥ 1.5	Python daemon process library

[1]These names match the corresponding Debian/Ubuntu packages.
[2]Python 3.x support would require some minor code updates throughout.
[3]Not needed if building from a `.tar.gz` package.

Table 3-3: Software requirements for building and running NilmTools.

3.1.3 NilmTools

The NilmTools package contains a variety of programs, and their installation is similar to the installation of NilmDB described in Section 3.1.1. NilmTools depends on software components provided by NilmDB, and so NilmDB must be installed on the local system first. However, the programs that comprise NilmTools can connect to remote systems, and so it is not necessary to *run* NilmDB on the same system in order to use the tools.

3.1.3.1 Building and Installation

Table 3-3 lists the external software dependencies for NilmTools. Many dependencies are shared between NilmDB. After installing NilmDB, the additional packages required by NilmTools can be installed as follows:

```
# Install additional NilmTools requirements
sudo apt-get install python-numpy python-scipy python-daemon
```

Once dependencies are installed, the source code for NilmTools can be retrieved via git. The following commands will clone the repository and check out a specific tagged revision:

```
# Clone the repository into user's home directory
cd
git clone https://git.jim.sh/jim/lees/nilmtools.git
cd nilmtools

# Check out a specific revision (optional; omit for latest code)
git checkout nilmtools-1.3.3
```

The NilmTools software can then be built and installed system-wide with:

```
# Install
sudo make install
```

At this point, the following programs have been installed to `/usr/local/bin` and should be executable at the command line: `nilm-decimate`, `nilm-decimate-auto`, `nilm-insert`, `nilm-copy`, `nilm-copy-wildcard`, `nilm-sinefit`, `nilm-prep`, `nilm-cleanup`, `nilm-median`, `nilm-trainola`, and `nilm-pipewatch`. Their usage is detailed in Section 3.4.2.

3.1.4 Automated Acquisition and Processing

There are many ways that data in NilmDB can be acquired and processed, and there are no fixed configurations that are suitable for all situations. External software is required to perform the data capture, and the network-aware nature of NilmDB allows for complex setups where different systems may be used for acquisition, data store, and processing. In general, a higher-level software tool like the NILM Manager (Section 2.4) is expected to orchestrate the individual programs and systems that perform these actions.

Nevertheless, a simple basic setup is outlined here that mirrors a typical NILM processing workflow [4, 6, 7, 13, 15, 16, 20, 24, 44]. This example assumes a LabJack or NerdJack capture card, readable with the `ethstream` software [45, 46], configured on a three-phase system such that the first three channels are capturing φA, φB, and φC currents, and the next three channels are capturing φA, φB, and φC voltages. The

processing will be controlled with the `cron` job scheduler, and the following steps will be performed:

- Acquire raw data continuously to the `/data/raw` stream (using `ethstream`, `nilm-insert`, and `nilm-pipewatch`).
- Find zero crossings for φA voltage and save to `/data/sinefit` (using `nilm-sinefit`).
- Calculate spectral envelopes for φA, φB, and φC current and save to `/data/prep-a`, `/data/prep-b`, and `/data/prep-c` (using `nilm-prep` with appropriate added phase shift).
- Decimate the raw and preprocessed data streams for plotting (using `nilm-decimate-auto`).
- Clean up old data, saving up to two weeks of raw data and one year of preprocessed data (using `nilm-cleanup`).

3.1.4.1 Building and Installation

The following is a very brief guide to installing `ethstream` and other needed dependencies:

```
# Install build dependencies, and utilities used by the cron job
sudo apt-get install build-essential help2man moreutils

# Clone the ethstream repository into user's home directory
cd
git clone https://git.jim.sh/jim/lees/ethstream.git
cd ethstream

# Check out a specific revision (optional; omit for latest code)
git checkout ethstream-1.3

# Build and install ethstream
make
sudo make install
```

3.1.4.2 Configuration and Scripts

A series of scripts and configuration files should be created. Capture is controlled by the script `/home/nilmdb/capture.sh`, containing:

```
#!/bin/bash -e
nilm-pipewatch --daemon --lock "/tmp/nilmdb-capture.lock" --timeout 30 \
  "ethstream -a 192.168.1.209 -n 6 -r 8000" \
  "nilm-insert -m 10 -r 8000 --live /data/raw"
```

Processing is controlled by the script `/home/nilmdb/process.sh`, containing:

```
#!/bin/bash -e

# The following ensures that only one copy of this script runs at a time:
LOCKFILE=/tmp/nilmdb-process.lock
exec 99>$LOCKFILE
flock -n -x 99 || exit 0
trap "rm -f $LOCKFILE" 0

# Perform desired processing steps
nilm-sinefit -c 4 /data/raw /data/sinefit
nilm-prep -c 1 -r 0   /data/raw /data/sinefit /data/prep-a
nilm-prep -c 2 -r 120 /data/raw /data/sinefit /data/prep-b
nilm-prep -c 3 -r 240 /data/raw /data/sinefit /data/prep-c
nilm-decimate-auto /data/raw "/data/prep*"
nilm-cleanup --yes /home/nilmdb/cleanup.cfg
```

Cleanup is controlled by the configuration file `/home/nilmdb/cleanup.cfg`, containing:

```
[/data/raw]
keep = 2w

[/data/sinefit]
keep = 1y
decimated = false

[/data/prep-*]
keep = 1y
```

The following shell commands can then be executed to make the appropriate scripts executable, as well as create the required streams in NilmDB:

```
# Mark scripts executable
chmod +x /home/nilmdb/capture.sh
chmod +x /home/nilmdb/process.sh

# Create streams
nilmtool create /data/raw uint16_6
nilmtool create /data/sinefit float32_3
nilmtool create /data/prep-a float32_8
nilmtool create /data/prep-b float32_8
nilmtool create /data/prep-c float32_8
```

3.1.4.3 Testing and Final Steps

The raw capture process can be manually started by executing the command:

```
sudo -u nilmdb /home/nilmdb/capture.sh
```

To verify that it is running, use `nilmtool` to look at the data extents and verify that they are changing. For example,

```
nilmtool list --ext /data/raw
```

The processing steps can be run manually with:

```
sudo -u nilmdb /home/nilmdb/process.sh
```

The output of the proessing can be used to verify that it ran correctly.

Finally, the `cron` jobs can be created to call these scripts automatically. Since the scripts should run as the `nilmdb` user, the `crontab` should be set up under that user. It can be edited interactively by running:

```
sudo -u nilmdb crontab -e
```

and then adding the following lines at the end of the file:

```
SHELL=/bin/bash
PATH=/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Run NilmDB capture and processing scripts every 5 minutes
*/5 * * * * chronic /home/nilmdb/capture.sh
*/5 * * * * chronic /home/nilmdb/process.sh

# Check NilmDB database for errors at startup, and fix them
@reboot chronic nilmdb-fsck --fix --no-data /home/nilmdb/db
```

The last line ensures that potential database errors due to improper shutdown are corrected at boot. The database path should be adjusted accordingly. Note that this relies on the database not being locked immediately by the web server; while this works in most situations, more complex setups may wish to integrate the `nilmdb-fsck` call with system startup in a way that guarantees its execution before any other process that might use NilmDB.

3.2 NilmDB Interfaces

The HTTP API, Python API, and command-line interfaces to the main NilmDB database server are described in the following sections.

3.2.1 HTTP API

Interaction with the NilmDB server takes place through a standard HTTP/1.1 compliant interface [36]. Per the HTTP specification, this interface is a request/response architecture supporting persistent connections, pipelining, chunked transfers, and identification of character sets and content types. In order to support direct access from Javascript running in a web browser, the API additionally supports the Cross Origin Request Sharing (CORS) mechanism [47].

Method	Usage
`GET` `HEAD`	Retrieve information and data non-destructively; actions such as listing streams or reading data. Requests with these methods should not trigger any changes to the database. Query parameters are provided in the request URL.
`POST`	Perform actions that may change the database; actions such as creating streams or deleting data. Query parameters are provided in the request body.
`PUT`	Data upload for inserting data into a stream. Query parameters are provided in the request URL, and bulk data is provided as in the request body.

Table 3-4: HTTP API request methods, uses, and parameter locations.

HTTP defines methods that perform actions on particular resources. The resources are identified by Uniform Resource Locator (URL) and the requests can contain query parameters. The NilmDB API supports methods `GET`, `HEAD`, `POST`, and `PUT`. The use cases for each method are summarized in Table 3-4, as well as the expected location of query parameters.

The API for NilmDB consists of a series of resources that are accessed by Uniform Resource Identifier (URI). For example, the resource for retrieving a list of streams in the database is `/stream/list`. The base URL of the NilmDB installation is prepended to this URI to build a complete request URL `http://<host>/nilmdb/stream/list`. Query parameters are supplied along with the request; examples of such parameters include time ranges and stream paths.

For requests where query parameters are located in the URL, the standard URL-escaped query string format should be used [48], encoded in the UTF-8 character set before escaping. For `POST` requests, query parameters may be encoded into the standard `application/x-www-form-urlencoded` format, or alternately presented as a dictionary in Javascript Object Notation (JSON) format [35] with content type `application/json`. The API call documentation that follows includes a list of supported parameters for each particular URI.

Server responses can be returned in multiple formats. Most API calls return JSON with content type `application/json`, with the specific data structure defined by the action being requested. Some API calls that return a large quantity of data, such as `/stream/intervals`, respond with a chunked encoding that includes multiple JSON objects separated by carriage-return / linefeed (CRLF) pairs. The non-standard content type `application/x-json-stream` is used for such responses. The API documentation includes the response content type and data format for each call.

Several request and response parameters incorporate timestamps, which are incorporated into the intervals and stored data. By convention, these timestamps usually represent absolute time measured in microseconds since the Unix time epoch (1970/01/01 00:00 UTC), but the interface supports arbitrary 64-bit signed integer values. One variation may be to use the timestamps to denote microseconds since some alternate reference time. The timestamps are further described in Section 2.2.3.

Errors are indicated with standard HTTP response codes such as `405 Method Not Allowed` or `400 Bad Request`. Error bodies are JSON dictionaries that contain additional information about the error. Within these bodies, the `message` key will typically contain a human-readable error, and the `traceback` key may contain a Python traceback if the server did not gracefully handle the error.

In the example API transactions that follow, requests to the server are denoted with a light blue line in the margin, and responses are denoted with a dark green line. For JSON responses, whitespace may be added for readability. Only relevant headers are included in both the request and response; in the actual transactions with the server, standard HTTP headers such as `Content-Length` and `Host` are required.

3.2.1.1 `/` – NilmDB short status text

Synopsis

```
GET /
```

Description

Returns a human-readable string with the NilmDB version number and local hostname. This is intended to assist with configuration and testing during installation, and is the default page that will be returned if a user visits the NilmDB base URL with a web browser.

Request Parameters

None

Response

Returns a `text/plain` string of human-readable text.

Example

```
GET / HTTP/1.1

HTTP/1.1 200 OK
Content-type: text/plain;charset=utf-8

This is NilmDB version 1.9.7, running on host pilot.lees.
```

3.2.1.2 `/version` – Get NilmDB server version

Synopsis

```
GET /version
```

Description

Returns the NilmDB server version number in JSON format.

Request Parameters

None

Response

Returns a JSON object containing a single string. The version number follows typical software version number conventions (i. e., 1.8.10 is newer than 1.8.9).

Example

```
GET /version HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

"1.9.7"
```

3.2.1.3 `/dbinfo` – Get database information

Synopsis

```
GET /dbinfo
```

Description

Returns information about the server's database storage subsystem, including the local filesystem path and disk space usage.

Request Parameters

None

Response

Returns a JSON dictionary. The keys of this dictionary and their values are shown in Table 3-5. Note that the "filesystem" here refers to the particular filesystem on the server on which the NilmDB database is stored, and does not include any other filesystems which may be present on the server.

Key	Value
`path`	Filesystem path where the NilmDB database is stored.
`size`	Total size of the NilmDB database, in bytes.
`other`	Size of non-NilmDB data on the same filesystem, in bytes.
`reserved`	Reserved space on the same filesystem, in bytes.
`free`	Free space on the same filesystem, in bytes.

Table 3-5: Contents of the `/dbinfo` HTTP response.

From this, the total usable size of the disk can be calculated as: `total` = `size` + `other` + `free`.

Example

```
GET /dbinfo HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

{
  "path": "/home/nilmdb/db",
  "other": 411695028624,
  "reserved": 7370223616,
  "free": 164507668480,
  "size": 143174059632
}
```

3.2.1.4 `/stream/list` – List streams in the database

Synopsis

```
GET /stream/list
GET /stream/list?layout=float32_8
GET /stream/list?path=%2Fnewton%2Fprep&extended=1
```

Description

Lists the streams in the database and their layout types. With the `extended=1` parameter, returned data will also include summary information about the data in the stream, such as the maximum extent of interval timestamps and the total amount of data present.

Request Parameters

Parameters are shown in Table 3-6. Required parameters are marked with $*$. Wildcards are not supported in `path` or `layout`.

*	Name	Description
	`path`	Stream path about which to return information, such as "/sharon/raw". If not specified, consider all paths.
	`layout`	Layout types to return information about, such as `uint16_6`. If not specified, consider all layouts.
	`extended`	Whether to return extended information. Use 0 or leave unspecified to return just path and layout, or specify as 1 to return extended info.

Table 3-6: Parameters for the `/stream/list` HTTP request.

Response

Information is returned for paths that match `path` and `layout` exactly. The format of the response is nested JSON list of lists. The outer list has one entry for each returned path:

```
[ <pathentry>, <pathentry>, ... ]
```

Each *<pathentry>* is another list with elements shown in Table 3-7. If request parameter `extended` was zero or unspecified, the list contains only the first two entries, otherwise, all six entries are included. Additional entries within the *<pathentry>* list may be included in future NilmDB server versions and should be ignored.

Performance Notes

Retrieving extended stream information for a stream with m intervals is an $O(m)$ operation and does not require querying the bulkdata backend. Since streams will have relatively few intervals compared to the amount of data stored, this should be fast.

Entry	Example	Description
1	`/sharon/raw`	Path name of this stream, following the structure described in Section 2.2.1.
2	`uint16_6`	Data type of the stream, following the format described in Section 2.2.3.
3	`1359950402000000`	The earliest start time of any interval in the stream, in microseconds since epoch, or "null" if no intervals.
4	`1359950639911250`	The latest end time of any interval in the stream, in microseconds since epoch, or "null" if no intervals.
5	`1903290`	Total rows of data in this stream.
6	`237911250`	Total amount of time covered by the intervals of this stream, in microseconds. Gaps between intervals would cause this number to be less than the difference between the "start time" and "end time" in fields 3 and 4.

Table 3-7: List entries for one path in the `/stream/list` HTTP response.

Example

```
GET /stream/list?layout=float32_8&extended=1 HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

[ [ "/bp/startup/prep-a", "float32_8",
  1305297300000000, 1305298802028251, 46616, 1502028251 ],
  [ "/bp/startup/prep-b", "float32_8",
  1305297300000000, 1305298802028251, 46616, 1502028251 ],
  [ "/bp/startup/prep-c", "float32_8",
  1305297300000000, 1305298802028251, 46616, 1502028251 ],
  [ "/lees-compressor/no-leak/prep", "float32_8",
  1360017784000000, 1361579632617001, 21516735, 359088115267 ],
  [ "/test/prep", "float32_8", null, null, 0, 0 ]
]
```

3.2.1.5 `/stream/create` – Create a new stream

Synopsis

```
POST /stream/create
{ "path": "/newton/prep", "layout": "float32_8" }
```

Description

Create a new empty stream in the database.

Request Parameters

Parameters are shown in Table 3-8. Required parameters are marked with ∗.

∗	Name	Description
∗	`path`	Stream path to create, following the structure described in Section 2.2.1 and below.
∗	`layout`	Data type of the stream, following the format described in 2.2.3 and below.

Table 3-8: Parameters for the `/stream/create` HTTP request.

Stream paths consist of N elements separated by `/` characters. They can be viewed as a tree-like structure similar to filesystem locations, with the first $N - 1$ elements of the path acting like directories and the final element acting as a filename. All paths must have at least two elements, and no stream can be created such that it would be a child or parent of an existing stream. For example, given an existing stream `/my/fun/experiment`, invalid stream paths include `/my/fun` and `/my/fun/experiment/work`. Valid paths include `/my/fun/stuff`, `/my/tedious/experiment`, and `/my/work`.

Layout types are of the form `<type>_<count>`. The `<type>` is one of those described in Section 2.2.3, such as `uint16`, `int64`, or `float32`. `<count>` is a numeric count of how many data elements there are, per row. Streams store rows of homogeneous data only, and the largest supported `<count>` is 1024. Generally, counts should fall within a much lower range, typically between 1 and 32.

Response

On success, `200 OK` is returned, and the body is a JSON object containing "null".

Errors

Invalid `path` or `layout` parameters, or paths that cannot be created due to conflict with other stream paths, result in a `400 Bad Request` response.

Examples

```
POST /stream/create HTTP/1.1
Content-Type: application/json

{ "path": "/asdf/qwer", "layout": "bad_1" }
HTTP/1.1 400 Bad Request
Content-type: application/json

{ "status": "400 Bad Request",
  "message": "no such layout: bad data type",
  "traceback": "" }
```

```
POST /stream/create HTTP/1.1
Content-Type: application/json

{ "path": "/newton/prep", "layout": "float32_8" }
HTTP/1.1 200 OK
Content-type: application/json

null
```

3.2.1.6 `/stream/destroy` – Fully remove an empty stream

Synopsis

```
POST /stream/destroy
{ "path": "/newton/prep" }
```

Description

Fully remove a stream and associated metadata from the database. All existing intervals must been removed with `/stream/remove` first (Section 3.2.1.12).

Request Parameters

Parameters are shown in Table 3-9. Required parameters are marked with ∗.

∗	Name	Description
∗	`path`	Stream path to destroy.

Table 3-9: Parameters for the `/stream/destroy` HTTP request.

Response

On success, `200 OK` is returned, and the body is a JSON object containing "null".

Errors

If `path` does not exist, `404 Not Found` is returned. If the stream still has intervals as listed by `/stream/list` (Section 3.2.1.4), `400 Bad Request` is returned.

Performance Notes

Since all data must have been previously removed, this operation is fast.

Examples

```
POST /stream/destroy HTTP/1.1
Content-Type: application/json

{ "path": "/old/stream" }
HTTP/1.1 200 OK
Content-type: application/json

null
```

3.2.1.7 `/stream/rename` – Rename a stream

Synopsis

```
POST /stream/rename
{ "oldpath": "/compressor/raw",
  "newpath": "/compressor/no-leak/raw" }
```

Description

Rename or relocate a stream in the database from one path to another. Metadata and intervals, if any, are relocated to the new path name.

Request Parameters

Parameters are shown in Table 3-10. Required parameters are marked with ∗.

∗	Name	Description
∗	`oldpath`	Existing stream path.
∗	`newpath`	New stream path.

Table 3-10: Parameters for the `/stream/rename` HTTP request.

The new stream path must follow the standard guidelines as described in the documentation for `/stream/create` (Table 3-8). Note that it is acceptable for `newpath` to be a parent or child of `oldpath`, as long as `newpath` does not conflict with any other existing streams according to the standard guidelines. For example, it is acceptable to rename `/my/fun/experiment` to `/no/fun`.

Response

On successful rename, `200 OK` is returned, and the body is a JSON object containing "null".

Errors

If `oldpath` does not exist, `404 Not Found` is returned. If `newpath` is invalid, a `400 Bad Request` response is returned. In all error cases, the stream will still exist at `oldpath`.

Notes

Metadata *contents* are not changed by this operation. Any software tools that store and use path names stored in metadata keys or values will need to update them accordingly.

Examples

```
POST /stream/rename HTTP/1.1
Content-Type: application/json

{ "newpath": "/newton/prep",
  "oldpath": "/totally/different/thing" }
HTTP/1.1 200 OK
Content-type: application/json

null
```

3.2.1.8 `/stream/intervals` – List intervals within a stream

Synopsis

```
GET /stream/intervals?path=/cold/filtered
GET /stream/intervals?path=/cold/filtered&diffpath=/something/else
```

Description

List the intervals present in the given stream, optionally limiting and truncating the returned intervals to a given time range.

If `diffpath` is specified, the server calculates and returns the set-difference between the intervals in `path` and `diffpath`; that is, only interval ranges that are present in `path` and not present in `diffpath` are returned.

Mathematically, let the interval set present in `path` be denoted P, the interval set in `diffpath` be denoted D, and the interval set equivalent to the requested time range be denoted T. Then the returned set is $P \cap D^c \cap T$.

For example, if `path` contains one interval $[100 \rightarrow 300\rangle$, `diffpath` contains one interval $[0 \rightarrow 200\rangle$, and the start and end times are unspecified, the result is the single interval $[200 \rightarrow 300\rangle$.

Request Parameters

Parameters are shown in Table 3-11. Required parameters are marked with $*$.

*	Name	Description
*	`path`	Stream path for which to return intervals.
	`diffpath`	Omit from the response any interval ranges that are also present in `diffpath`.
	`start`	Start time, in microseconds since epoch. If specified, limit response to ranges that begin at or after `start`.
	`end`	End time, in microseconds since epoch. If specified, limit response to ranges that end at or before `end`.

Table 3-11: Parameters for the `/stream/intervals` HTTP request.

Response

Returned intervals are always non-overlapping and sorted in ascending order by start time.

On success, `200 OK` is returned, and the matching intervals are streamed in a chunked `application/x-json-stream` response. Each interval is returned as an independent JSON list `[<start>, <end>]` followed by CRLF.

Errors

If `path` or `diffpath` does not exist, `404 Not Found` is returned.

Performance Notes

If `path` and `diffpath` contain m and n intervals, respectively, then this operation will take $O(m+n)$ time. If there are a large number of intervals to return, the server may choose to transparently subdivide the request to the backend server to prevent excessive blocking, which may cause short delays in the chunked JSON responses.

Example

```
GET /stream/intervals?path=%2Ffoo%2Ftest&start=0&end=2000 HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/x-json-stream
Transfer-Encoding: chunked

4a
[100, 145]
[175, 200]
[300, 350]
[400, 450]
[500, 550]
[1000, 1050]

38
[1100, 1101]
[1199, 1250]
[1400, 1450]
[1500, 1550]

0
```

3.2.1.9 `/stream/set_metadata` – Set metadata for stream

Synopsis

```
POST /stream/set_metadata
{ "path": "/data/raw",
  "data": { "key1": "value1", "key2": "value2" }
}
```

Description

Set arbitrary key/value metadata associated with a particular stream path. This action will overwrite *all* existing metadata for this path with the provided values. Use `/stream/update_metadata` (Section 3.2.1.11) to modify or insert keys without affecting others.

Keys and values are both arbitrary Unicode text strings. Keys must be unique.

Request Parameters

Parameters are shown in Table 3-12. Required parameters are marked with ∗.

∗	Name	Description
∗	`path`	Stream path for which to set metadata.
∗	`data`	JSON dictionary containing key/value pairs. Keys and values must both be strings.

Table 3-12: Parameters for the `/stream/set_metadata` HTTP request.

Note that the `data` parameter is always a JSON-encoded dictionary. If the content-type of the request is `application/json`, this dictionary can be included directly in the request body:

```
{ "path": "/test/raw", "data": { "k": "v" } }
```

If the content-type of the request is `application/x-www-form-urlencoded`, then the `data` parameter requires an extra encoding as JSON before the URL encoding step. For example, metadata consisting of key `k` and value `v` would be first encoded into JSON as

```
{"k":"v"}
```

and subsequently URL-encoded in the request body as:

```
path=%2Ftest%2Fraw&data=%7B%22k%22%3A%22v%22%7D
```

Response

On success, `200 OK` is returned, and the body is a JSON object containing "null".

Errors

If `path` does not exist, the response is `404 Not Found`. If the `data` parameter cannot be parsed, or if there is another formatting or data type problem with the keys or values, `400 Bad Request` is returned, and the `message` field in the response body may contain more information about the error.

Performance Notes

Keys and values are stored directly within the SQL database (Section 2.3.3), and operations such as `/stream/get_metadata` and `/stream/update_metadata` retrieve all data for a given stream from the database. As such, storing very large quantities of data is not recommended.

Example

```
POST /stream/set_metadata HTTP/1.1
Content-type: application/json

{ "path": "/test/greek",
  "data": { "alpha": "\u03b1", "beta": "\u03b2" } }
HTTP/1.1 200 OK
Content-type: application/json

null
```

3.2.1.10 `/stream/get_metadata` – Retrieve metadata

Synopsis

```
GET /stream/get_metadata?path=%2Fdata%2Fraw
GET /stream/get_metadata?path=%2Fdata%2Fraw&key=k1&key=k2
```

Description

Retrieve the key/value metadata pairs associated with a particular stream. If optional key parameters are specified, only metadata matching the given keys is returned. Keys and values are both arbitrary Unicode text strings.

Request Parameters

Parameters are shown in Table 3-13. Required parameters are marked with $*$.

*	Name	Description
*	`path`	Stream path for which to get metadata.
	`key`	Key names to retrieve. May be repeated, in which case multiple matching keys may be returned. If unspecified, all keys are returned.

Table 3-13: Parameters for the `/stream/get_metadata` HTTP request.

Response

On success, `200 OK` is returned, and the body is a JSON dictionary with the key/value pairs. If keys were specifically requested by the `key` parameter but are not present in the database, they are returned in the JSON dictionary with a value of `null`.

Errors

If `path` does not exist, the response is `404 Not Found`.

Example

```
GET /stream/get_metadata?path=%2Fdata%2Fraw&key=alpha&key=beta HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

{ "alpha": "\u03b1", "beta": null }
```

3.2.1.11 `/stream/update_metadata` – Update metadata

Synopsis

```
POST /stream/update_metadata
{ "path": "/data/raw",
  "data": { "key1": "value1", "key2": "value2" }
}
```

Description

Update the key/value metadata associated with a particular stream path. The given values overwrite existing values in the database, if any. Any keys that are present in the database, but not present in the parameters to this call, are left unchanged.

Deleting metadata entries is not explicitly supported, but values can be set to empty strings. To completely remove a key from the database instead, use `/stream/get_metadata` to retrieve the metadata, then set that metadata again with `/stream/set_metadata`, with the desired key excluded.

Request Parameters

Parameters are shown in Table 3-14. Required parameters are marked with ∗.

∗	Name	Description
∗	`path`	Stream path for which to update metadata.
∗	`data`	JSON dictionary containing key/value pairs. Keys and values must both be strings.

Table 3-14: Parameters for the `/stream/update_metadata` HTTP request.

As with `/stream/set_metadata`, the `data` parameter is always a JSON dictionary, regardless of request content type. See Section 3.2.1.9 for details.

Response

On success, `200 OK` is returned, and the body is a JSON object containing "null".

Errors

If `path` does not exist, the response is `404 Not Found`. If the `data` parameter cannot be parsed, or if there is another formatting or data type problem with the keys or values, `400 Bad Request` is returned, and the `message` field in the response body may contain more information about the error.

Example

```
POST /stream/update_metadata HTTP/1.1
Content-type: application/json

{ "path": "/test/greek",
  "data": { "alpha": "a", "gamma": "g" } }
HTTP/1.1 200 OK
Content-type: application/json

null
```

3.2.1.12 `/stream/remove` – Remove data from a stream

Synopsis

```
POST /stream/remove
{ "path": "/reactor/core",
  "start": 1332497210000000, "end": 1332497030000000 }
```

Description

Remove data from the specified time interval in the stream. This function will delete any stored data with timestamp t satisfying `start` $\leq t <$ `end`, and also remove the same range from the stream's interval list. If the requested range does not exactly match an existing interval, the stream's intervals will be truncated or split as necessary.

Request Parameters

Parameters are shown in Table 3-15. Required parameters are marked with $*$.

$*$	Name	Description
$*$	`path`	Stream path for which to remove data.
	`start`	Start time, in microseconds since epoch. Default is $-\infty$ if not specified.
	`end`	End time, in microseconds since epoch. Default is $+\infty$ if not specified.

Table 3-15: Parameters for the `/stream/remove` HTTP request.

Note that omitting `start` and `end` will result in all data being removed from this stream. Since this is a destructive operation, it is recommended that `start` and `end` are always specified explicitly.

Response

On success, `200 OK` is returned, along with the count of how many data rows were removed. The removal operation is potentially slow as data is deleted from disk storage. In order to keep the HTTP connection alive, the server sends a chunked `application/x-json-stream` response, where each transmitted line periodically reports how many data rows have been removed since the previous line. At the end of the transfer, the total number of rows of data removed from the database is equal to the sum of all of these numbers.

Errors

If `path` does not exist, `404 Not Found` is returned. If `start` or `end` are invalid, `400 Bad Request` is returned.

Performance Notes

Removing a large amount of data can take a while, due to the unavoidable time it takes to actually delete the data files from low level storage from disk. Accordingly, file removal is done in chunks, and there is an internal maximum number of rows that can be removed per chunk. Each line of the streaming response corresponds to one of those chunk, and other pending API calls may be handled in between them. Therefore, the total amount of time it takes to remove data can be somewhat unpredictable. The progress could be estimated by first counting the number of rows using `/stream/extract` (Section 3.2.1.14) and then comparing the running sum of removed rows to that total.

It is important to avoid removing a large number of scattered, small intervals of data. At the storage backend, data is split across multiple small files. As contiguous intervals of data are removed, regions of those files are marked as

being unused. However, disk space might not be reclaimed until the unused portions have grown large enough, or entire files have been marked unused. This means that certain uncommon patterns of data removal may have little or no effect on the available disk space, and performance may degrade significantly as intervals become fragmented. For example, deleting every other row in a stream would yield the worst case performance.

If such operations are desired, it would be more efficient to create a new stream and selectively copy the desired data into it with `/stream/extract` and `/stream/insert`. Similarly, a stream that has previously suffered degraded performance due to this type of workload could be "defragmented" by creating a new stream, copying all data to it, destroying the old stream, and renaming the new stream to the old path.

Example

```
POST /stream/remove HTTP/1.1

{ "path": "/sharon/prep",
  "start": 1332490000000000, "end": 13324970303133700 }
HTTP/1.1 200 OK
Content-type: application/x-json-stream
Transfer-Encoding: chunked

9
1048576

9
1048576

9
1048576

7
15600

0
```

3.2.1.13 `/stream/insert` – Insert a new interval of data

Synopsis

```
PUT /stream/insert?path=%2Fdata%2Fraw&start=1234&end=6789
PUT /stream/insert?path=%2Fdata%2Fraw&start=1234&end=6789&binary=1
```

Description

Insert new data into a stream. A new interval is created from `start` to `end`, and the provided rows of timestamped data are inserted into this interval. Each timestamp t must satisfy `start` $\leq t <$ `end`, and the new interval must not overlap any existing intervals in the stream.

Request Parameters

Parameters are shown in Table 3-16. Required parameters are marked with $*$.

*	Name	Description
*	`path`	Existing stream path into which to insert data
*	`start`	Start time of the newly created interval, in microseconds since epoch
*	`end`	End time of the newly created interval, in microseconds since epoch
	`binary`	If specified as `1`, uploaded data is in binary format, otherwise ASCII-formatted text. Both formats are described in the section "Request Body" below.

Table 3-16: Parameters for the `/stream/insert` HTTP request.

A new interval [`start` $\rightarrow$ `end`$\rangle$ is created within the stream at `path` and populated with the data provided in the request body.

Request Body

Within an interval, data is stored in rows. The rows generally correspond to a single data point or event, and contain a timestamp and some number of

homogeneous columns of data, as determined by the layout specified when the stream was created. Layout and data types are described in Section 2.2.3.

Timestamps are arbitrary signed 64 bit integer values, ranging from $-2^{63} \rightarrow 2^{63}-1$. For compatibility with software such as Javascript that may store integer values as IEEE 754 double-precision floating point numbers, limiting timestamps to the range $-2^{53} \rightarrow 2^{53}$ is suggested. Within an interval, timestamps are monotonic and fall within the half-open interval bounds [`start` → `end`⟩. NilmDB itself does not interpret timestamps, but many of the supporting tools and filters (Sections 3.4.1 and 3.4.2) treat them as the number of microseconds elapsed since January 1, 1970 at 00:00:00 UTC.

The specific format of the request body varies based on whether `binary` was specified. The two variants are described below.

Request Body (Text Format)

If the `binary` parameter is zero or unspecified, rows are provided as human-readable ASCII text, and the content-type of the document body should be `text/plain`. Integer numbers are formatted according to the requirements of C11 standard function call `strtol` [49], with a fixed base of 10. Floating-point numbers are formatted according to `strtod` [49], which supports exponential notation and special values such as `infinity` and `NaN`. Thus, `12345`, `5e6`, `-10.331e+05`, and `-inf` are all valid.

The text format consists of a sequence of rows of data. Given the stream layout `<type>_<count>`, a single row of text input is formatted as:

- A timestamp, either as an signed 64-bit integer or floating point value. Floating point values are rounded to the nearest 64-bit integer.
- Exactly `<count>` numbers of type `<type>`, separated from the timestamp and each other by any number of spaces.
- A single newline character (`\n`).

Type	Bytes	Format
`int8`	1	Two's complement signed integer
`uint8`	1	Unsigned integer
`int16`	2	Two's complement signed integer
`uint16`	2	Unsigned integer
`int32`	4	Two's complement signed integer
`uint32`	4	Unsigned integer
`int64`	8	Two's complement signed integer
`uint64`	8	Unsigned integer
`float32`	4	IEEE 754 single-precision floating point [34]
`float64`	8	IEEE 754 double-precision floating point [34]

Table 3-17: Binary data formats for each layout type. Multi-byte values are stored and transferred in little-endian order.

Additionally, the input text may contain comments, which are ignored. Comments start with a `#` character and extend to the next newline character. Comments may appear before any numbers on a line, in which case the entire line is ignored, or may appear at the end of the line, after the timestamp and all `<count>` values.

An example of text formatted data is shown in the first example below.

Request Body (Binary Format)

If the `binary` parameter is `1`, rows are provided in a compact, fixed-length binary representation, and the content-type of the document must be `application/octet-stream`. Given the stream layout `<type>_<count>`, each row is transmitted as a single record containing one timestamp of type `int64` and `<count>` values of type `<type>`. All multi-byte numbers are transmitted in little-endian order. The specific format for each of the supported layout types is shown in Table 3-17.

For example, a single row of type `float32_3` would be formatted in binary as a sequence of 20 hexadecimal bytes: 8 bytes for the timestamp and 4 bytes for each of the three `float32` values.

Each row record is concatenated and sent directly in the request body. When inserting n rows of data into a stream, the length of the request body must be exactly $n * (8 + (c * b))$ bytes, where c is the layout `<count>` and b is the number of bytes corresponding to the layout `<type>`.

An example of binary formatted data is shown in the second example below.

Response

On success, `200 OK` is returned, and the body is a JSON object containing "null".

Performance Notes

Inserting text data is slower than binary data, due to the extra processing needed to convert from the text representation to integers or floating-point numbers. Binary data is also more compact.

The overall performance of NilmDB when inserting, extracting, or manipulating data in a stream is highly related to the number of intervals present. For a stream with n intervals, interval storage takes $\mathrm{O}\big(n\big)$ space, and interval lookup, addition, or removal takes takes $\mathrm{O}\big(\log(n)\big)$ time. Many NilmDB tools, such as the filters in Section 2.2.5, operate on individual intervals, and so keeping interval counts low will increase performance.

At the other extreme, intervals that are too large will incur costs when locating a particular timestamped data row for retrieval. However, such intervals are automatically split by the server, and so the user should focus on providing the widest intervals possible.

The NilmDB server applies a small optimization when two intervals are inserted back-to-back. Given an existing stream interval $[s_1 \to e_1\rangle$ and a new stream interval $[s_2 \to e_2\rangle$, the server will consolidate them into the single interval $[s_1 \to e_2\rangle$ if two conditions are met:

- $e_1 = s_2$.
- No other data was added to the stream between the adding of $[s_1 \to e_1\rangle$ and $[s_1 \to e_2\rangle$.

Note that the second condition means that this optimization will *not* be applied in the case where two processes are simultaneously writing data into the same stream, even if they are writing data in disjoint time regions. Within any given stream, it is always better to write data in a linear fashion.

If performance degradation is observed as a stream grows larger, the "performance health" can be judged by looking at the ratio of number of rows of data (reported by `/stream/extract`, Section 3.2.1.14) versus the number of intervals (reported by `/stream/intervals`, Section 3.2.1.8). An ideal ratio from a storage point of view is on the order of 10^6 or 10^7 rows per interval. If the number is significantly lower, performance could be increased by:

- "Filling in" any small gaps between non-contiguous intervals by using `/stream/insert` with no data.
- Copying each interval of data in-order to a new stream, which allows the consolidation optimization to take place.

Errors

Invalid paths return `404 Not Found`. A `400 Bad Request` response, with more detail in the error body, will be returned if any of the following error conditions occur:

- Invalid `path`, `start`, or `end` parameters.
- The interval described by `start` and `end` overlaps an existing interval in the stream.
- `binary` is specified, but the request content-type is not `application/octet-stream`.

- Uploaded data is not formatted correctly for the stream layout type, or data values are out of range.
- Uploaded data timestamps t are not monotonically increasing, or do not satisfy `start` $\leq t <$ `end`. Note that duplicated timestamps will also trigger this error, and all inserted data rows must have a unique timestamp within a given stream.

Additionally, `500 Internal Server Error` may be raised on I/O errors such as running out of disk space on the server.

Examples

In this first example, 8 rows of data are inserted to a stream of type `uint16_6`, using ASCII formatted text.

```
PUT /stream/insert?path=%2Fa%2Fb&start=1360017784000000& ↩ (continued)
                  end=1360017784001000 HTTP/1.1
Content-Type: text/plain
Content-Length: 424

1360017784000000 33056 33056 33168 33056 33056 33072
1360017784000125 33056 33072 33168 33024 33056 33088
1360017784000250 33072 33056 33168 33040 33072 33072
1360017784000375 33072 33072 33168 33040 33056 33072
1360017784000500 33056 33072 33184 33040 33056 33072
1360017784000625 33056 33056 33168 33040 33072 33072
1360017784000750 33056 33072 33184 33056 33056 33072
1360017784000875 33056 33088 33168 33040 33056 33072
HTTP/1.1 200 OK
Content-type: application/json

null
```

The second example is the same data in binary format. Binary data is represented here as a list of 2-digit hexadecimal values separated by spaces, and the timestamp portions of the binary data are underlined.

```
PUT /stream/insert?path=%2Fa%2Fb&start=1360017784000000& ↩ (continued)
                   end=1360017784001000&binary=1 HTTP/1.1
Content-Type: application/octet-stream
Content-Length: 160

00 4e e7 d0 ed d4 04 00 20 81 20 81 90 81 20 81
20 81 30 81 7d 4e e7 d0 ed d4 04 00 20 81 30 81
90 81 00 81 20 81 40 81 fa 4e e7 d0 ed d4 04 00
30 81 20 81 90 81 10 81 30 81 30 81 77 4f e7 d0
ed d4 04 00 30 81 30 81 90 81 10 81 20 81 30 81
f4 4f e7 d0 ed d4 04 00 20 81 30 81 a0 81 10 81
20 81 30 81 71 50 e7 d0 ed d4 04 00 20 81 20 81
90 81 10 81 30 81 30 81 ee 50 e7 d0 ed d4 04 00
20 81 30 81 a0 81 20 81 20 81 30 81 6b 51 e7 d0
ed d4 04 00 20 81 40 81 90 81 10 81 20 81 30 81
HTTP/1.1 200 OK
Content-type: application/json

null
```

The final example shows the typical format of details in the error response. Note that newlines and indentation have been added to the message string here, after `\n` sequences, to improve readability.

```
PUT /stream/insert?path=%2Fcontext%2Ftest&start=3000&end=4000 HTTP/1.1
Content-Type: text/plain
Content-Length: 35

3010 1.0 2.0 3.0 4.0
3020 2.0 3.0 4.0 5.0
3030 3.0 4.0 blah 6.0
3040 4.0 5.0 6.0 7.0
3050 5.0 6.0 7.0 8.0
HTTP/1.1 400 Bad Request
Content-type: application/json

{ "status": "400 Bad Request",
  "message":
    "error parsing input data: line 3, column 14: can't parse value\n
     3030 3.0 4.0 blah 6.0\n
                  ^\n",
  "traceback": "" }
```

3.2.1.14 `/stream/extract` – Extract data from a stream

Synopsis

```
GET /stream/extract?path=%2Fdata%2Fraw&start=1234&end=6789
GET /stream/extract?path=%2Fdata%2Fraw&start=1234&end=6789&binary=1
GET /stream/extract?path=%2Fdata%2Fraw&start=1234&end=6789&count=1
GET /stream/extract?path=%2Fdata%2Fraw&start=1234&end=6789&markup=1
```

Description

Extract data from a stream, or count the number of rows that would have been extracted. All data within the time range [`start` → `end`⟩ is included. This range does not need to match the intervals that were created when data was inserted into the stream; the request will be fulfilled using any available data.

Request Parameters

Parameters are shown in Table 3-18. Required parameters are marked with ∗.

∗	Name	Description
∗	`path`	Stream path from which to extract data
	`start`	Start time, in microseconds since epoch. Default is $-\infty$ if not specified.
	`end`	End time, in microseconds since epoch. Default is $+\infty$ if not specified.
	`binary`	If specified as `1`, extracted data is in binary format, otherwise ASCII-formatted text.
	`count`	If specified as `1`, return a count of data rows that match the request, instead of actual data.
	`markup`	If specified as `1`, extracted data will be marked with comments that delimit the extents of the intervals in the original data stream.

Table 3-18: Parameters for the `/stream/extract` HTTP request.

Note that omitting `start` and `end` will result in all data being returned from this stream. The `binary`, `count`, and `markup` options are mutually exclusive due to their incompatible response formats.

Response

On success, `200 OK` is returned, followed by data. The data is sent with the `chunked` transfer-encoding, and there is no set limit to how much data can be returned for a specific request.

If `count` was specified, the number of rows of data with timestamp t satisfying `start` $\leq t <$ `end` are returned as a single ASCII integer.

Otherwise, rows of data are returned. The NilmDB backend first locates the stream interval in which time `start` would be located, and further locates the first row of data in this interval with timestamp $t \geq$ `start`. This row, and all subsequent data in the interval, are returned as long as $t <$ `end`. If the stream interval ends, the next interval is considered, continuing until the timestamp exceeds `end` or there are no more intervals that could contain data in the requested time range.

The default is for data to be output as line-based ASCII-formatted text, analogous to the data format described for `/stream/insert` (Section 3.2.1.13), with content-type `text/plain`. Note that the text data is interpreted on insertion and reformatted on extraction, so the exact text contents may differ from the originally inserted data in terms of whitespace, comments, and number formats. Additionally, floating point precision limits may lead to rounding or truncation.

If `markup` is specified, the text output is additionally marked with information that denotes where the stream's internal intervals begin and end. These marks consist of comments of the form:

```
# <mark> <timestamp>
```

where `<mark>` is either the string `interval-start` or `interval-end`, and `<timestamp>` is a 64-bit integer. These marks convey the intervals that were present in the underlying stream, and can be used to help convey gaps in the underlying data. It is the same information that could be retrieved with a separate

`/stream/intervals` request (Section 3.2.1.8). Note that due to internal interval splitting and joining optimizations, these intervals may not match the exact intervals that were originally inserted into the database, but cover the same ranges of time. In particular, an `interval-end` mark may be followed by an immediate `interval-start` mark with the same timestamp, indicating no gap in the data.

If `binary` is specified, the response is instead formatted as raw binary data with content-type `application/octet-stream`. The format is as described for `/stream/insert` (Section 3.2.1.13 and Table 3-17). If data was inserted as binary, the extracted binary data will match the original data exactly.

To summarize, there are four distinct response types, based on the input parameters:

- Count only, as text (`count`)
- Text data
- Text data with markup (`markup`)
- Binary data (`binary`)

An example of each is provided in the example section below.

Errors

If `path` does not exist, `404 Not Found` is returned. If `start` or `end` are invalid, or the combination of parameters is invalid, `400 Bad Request` is returned. Note that a lack of data in the requested range will *not* return an error, but will return a successful empty response.

Performance Notes

Requesting `count` is a relatively fast operation as it only requires reading enough data to locate the rows at `start` and `end`. Client applications may wish to request a count before requesting the actual data in order to verify the number of rows

available, although this number may change between the two requests if data is actively being inserted or removed by another process.

Extracting text data is slower than binary data, due to the extra processing needed to convert from the on-disk representation to the ASCII-formatted text output. Binary data is also more compact.

For large requests, the server may internally break the request into multiple sub-requests. This is largely transparent but may result in short delays in the streaming output.

Example

These examples are all requesting the same region of data, which was inserted into the server as several intervals with a gap. This first example is extracting data in text mode:

```
GET /stream/extract?path=%2Fa%2Fb&start=1360017784000000& ↩ (continued)
                   end=1360017784001000 HTTP/1.1

HTTP/1.1 200 OK
Content-type: text/plain
Transfer-Encoding: chunked

1a8
1360017784000000 33056 33056 33168 33056 33056 33072
1360017784000125 33056 33072 33168 33024 33056 33088
1360017784000250 33072 33056 33168 33040 33072 33072
1360017784000375 33072 33072 33168 33040 33056 33072
1360017784000500 33056 33072 33184 33040 33056 33072
1360017784000625 33056 33056 33168 33040 33072 33072
1360017784000750 33056 33072 33184 33056 33056 33072
1360017784000875 33056 33088 33168 33040 33056 33072

0
```

The second example is extracting the count only:

```
GET /stream/extract?path=%2Fa%2Fb&start=1360017784000000& ↩ (continued)
                   end=1360017784001000&count=1 HTTP/1.1

HTTP/1.1 200 OK
Content-type: text/plain
Transfer-Encoding: chunked
```

```
2
8

0
```

This is extracting a slightly different time range, with interval markup:

```
GET /stream/extract?path=%2Fa%2Fb&start=1360017784000123& ↩ (continued)
                   end=1360017784000650&markup=1 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-type: text/plain
Transfer-Encoding: chunked

1cf
# interval-start 1360017784000123
1360017784000125 33056 33072 33168 33024 33056 33088
1360017784000250 33072 33056 33168 33040 33072 33072
# interval-end 1360017784000300
# interval-start 1360017784000375
1360017784000375 33072 33072 33168 33040 33056 33072
1360017784000500 33056 33072 33184 33040 33056 33072
# interval-end 1360017784000625
# interval-start 1360017784000625
1360017784000625 33056 33056 33168 33040 33072 33072
# interval-end 1360017784000650

0
```

Finally, the same data as the previous example, extracted in binary format. Binary data is represented here as a list of 2-digit hexadecimal values separated by spaces, and the timestamp portions of the binary data are underlined.

```
GET /stream/extract?path=%2Fa%2Fb&start=1360017784000123& ↩ (continued)
                   end=1360017784000650&binary=1 HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-type: application/octet-stream
Transfer-Encoding: chunked

64
7d 4e e7 d0 ed d4 04 00 20 81 30 81 90 81 00 81
20 81 40 81 fa 4e e7 d0 ed d4 04 00 30 81 20 81
90 81 10 81 30 81 30 81 77 4f e7 d0 ed d4 04 00
30 81 30 81 90 81 10 81 20 81 30 81 f4 4f e7 d0
ed d4 04 00 20 81 30 81 a0 81 10 81 20 81 30 81
71 50 e7 d0 ed d4 04 00 20 81 20 81 90 81 10 81
30 81 30 81

0
```

3.2.2 Python Client API

A complete Python client library is provided for accessing the functionality of the NilmDB server. This is the interface that is generally used by Python filters and supporting tools, like those in the NilmTools package (Section 3.4.1). The library is built and installed alongside NilmDB. It is not necessary to run the NilmDB server on a given system in order to use the client library on that system, but NilmDB must have been installed according to the procedure in Section 3.1.1.1.

Use of the client library is centered around the `Client` class provided in the module `nilmdb.client.client`. An object of type `Client` is instantiated with the URL of a NilmDB server, and provides a series of member functions that perform operations against the database by making HTTP calls. This module is described in Section 3.2.2.1.

Many of the functions in `Client` mirror the operations of the HTTP API, previously described in Section 3.2.1. In the following documentation, client library functions include references to the relevant HTTP API documentation where appropriate. Some `Client` functions, on the other hand, provide higher-level abstractions. A notable example is the `StreamInserter` context manager, returned by the `stream_insert` method, which provides a means for inserting data into the database while efficiently managing the intervals that get created. The methods of this class are described beginning on page 125.

The `NumpyClient` class in the module `nilmdb.client.numpyclient` extends `Client` by providing methods that allow for data to be manipulated as NumPy arrays. This is provided as a convenience for the user, as mathematical processing in Python often requires the use of NumPy. It also utilizes the binary data transfer functionality of the NilmDB server, and has significantly higher performance than the text-based `Client` for many workloads. This class is described in Section 3.2.2.2.

Within the client library functions, exceptions are used to indicate errors. The client library includes specific exceptions in the `nilmdb.client.errors` module that are typically used to indicate errors that arise during communication with the NilmDB

server. Other standard Python and NumPy exceptions may also be raised when appropriate. The error module is described in Section 3.2.2.3.

In addition to the client library, NilmDB provides several independent utility modules which may be useful to programs utilizing the client library. Two in particular, `nilmdb.utils.time` and `nilmdb.utils.interval`, are described in Section 3.2.2.4 and Section 3.2.2.5. Others can be found in the "Shared Utilities" source code in Appendix B.5.

Examples of how to use the NilmDB client library can be seen in the source code for NilmTools programs such as `nilm-cleanup` (Listing E-3), `nilm-trainola` (Listing E-11), and `nilm-insert` (Listing E-9).

3.2.2.1 Module `nilmdb.client.client`

This module contains the main **`Client`** class for communicating with a NilmDB server.

class `nilmdb.client.client.`**`Client`**(*url*)

Main client interface to NilmDB. *url* is the address of the NilmDB server, such as `"http://localhost/nilmdb/"`. The actual connection to the server may be deferred until the first method call that requires it.

This class can also be used as as a context manager, using the following structure:

```
with Client("http://localhost/nilmdb") as client:
    print client.version()
```

When used in this manner, the client, and any connection to the server, will be automatically closed when the `with` block exits, as if `client.close()` had been called.

Many methods in this class take *start* and *end* parameters. These are integers that typically represent microseconds since the Unix time epoch (1970/01/01 00:00 UTC). If specified as *None*, the times are interpreted as $-\infty$ and ∞, respectively.

close()

Close the connection; safe to call multiple times.

dbinfo() → *dict*

Return server database info as a dictionary. The dictionary includes information about the database location and free disk space; details on keys and values can be found in Table 3-5 on page 90.

geturl() → *string*

Return the base URL used by this client object.

stream_count(*path*,*start=None*,*end=None*) → *int*

Count and return the number of rows of data in the stream at *path* that fall in the interval [*start* → *end*⟩, without actually retrieving or transferring that data.

stream_create(*path*,*layout*)

Create a new stream at *path* with the given *layout*. For the format of *path* and *layout*, see the documentation of `/stream/create` in Section 3.2.1.5.

stream_destroy(*path*)

Fully delete a stream at the given *path* from the database. The stream must be empty; data can first be removed with with **stream_remove**.

stream_extract(*path*, *start=None*, *end=None*, *count=False*, *markup=False*, *binary=False*) → *generator*

Extract rows of data from *path* in the interval [*start* → *end*⟩. This function returns a generator, and iterating over the generator will yield the extracted data. For example:

```
for line in client.stream_extract("/test/foo"):
    print line
```

By default, the data is returned as ASCII formatted text and the generator will yield one line of text at a time. If *binary* is *True*, the generator will

yield arbitrarily sized chunks of raw binary data. The text and binary formats are described in detail in the documentation of `/stream/insert` in Section 3.2.1.13.

If *count* is *True*, the generator will yield a single line of text containing the number of rows of data present in the given interval, without actually fetching that data.

If *markup* is *True*, the ASCII formatted text will include comments marking where the stream's internal intervals begin and end. See the documentation of `/stream/extract` in Section 3.2.1.14 for details on the markup format.

stream_get_metadata(*path*, *keys=None*) → *dict*

Retrieve the key/value metadata pairs associated with the stream at *path*, returning it as a dictionary. If *keys* is provided as a list, only metadata matching the given keys is returned.

stream_insert_block(*path*, *data*, *start*, *end*, *binary=False*)

Insert data into a stream in a single operation. An interval $[start \rightarrow end\rangle$ is created at *path* and *data* is stored in that interval. The data is provided as text, or raw binary if *binary* is *True*.

In general, use **stream_insert_context** instead; this is a lower-level function that mirrors the functionality of `/stream/insert` in Section 3.2.1.13.

stream_insert_context(*path*, *start=None*, *end=None*) → **StreamInserter**

Insert data into a stream. Returns a **StreamInserter** context manager that allows data to be efficiently inserted into a stream in a piecewise manner. This is the primary function for inserting text-formatted data into NilmDB. This function may make multiple requests to the server, if the data is large enough or enough time has passed between insertions.

Data is provided to the context manager as ASCII text separated by newlines, and is aggregated and sent to the server in larger or smaller chunks as necessary. The format of the data must match the database layout for the given *path*. If *start* and *end* are not provided, they are deduced from the contents of the data by the context manager. See the **StreamInserter** documentation on page 125 for details on the context manager methods. A basic example of usage is:

```
with client.stream_insert_context("/test/foo") as ctx:
    ctx.insert("1234567000.0 1 2 3 4\n")
    ctx.insert("1234567001.0 4 3 2 1\n")
```

The text data format, as well as important performance notes regarding data intervals, is described in the documentation of `/stream/insert` in Section 3.2.1.13. Using this context manager instead of individual calls to **stream_insert_block** helps reduce the number of intervals created when the data is not all available at once, as the context manager can remain "open" while data is produced.

stream_insert(*path*, *data*, *start=None*, *end=None*)

Insert rows of ASCII text data into a stream. Use **stream_insert_context** instead; this is a compatibility function that is essentially:

```
with client.stream_insert_context(path, start, end) as ctx:
    ctx.insert(data)
```

stream_intervals(*path*, *start=None*, *end=None*, *diffpath=None*) → *generator*

List the intervals present in the stream at *path*, limited to the time range [*start* → *end*⟩. Returns a generator that yields a series of Python lists `[interval_start, interval_end]`. To get a complete nested list of all intervals, use e.g.:

```
all_intervals = list(client.stream_intervals(path))
```

If *diffpath* is specified, the server calculates and returns the set-difference between the intervals in *path* and *diffpath*; that is, only interval ranges

that are present in *path* and not present in *diffpath* are returned. See the documentation of `/stream/intervals` in Section 3.2.1.8 for details.

`stream_list`(*path=None*, *layout=None*, *extended=False*) → *generator*

List the streams in the database and their layout types. If *path* or *layout* are specified, only returns information about streams with the specified path or layout types.

Returns a generator that yields a series of Python lists. By default, these lists contain just `[path, layout]` for the matched streams. If *extended* is *True*, the lists contain additional extended information about the data in the stream, such as the maximum extent of interval timestamps and the total amount of data present. For full details on the entries in the returned list, see the documentation of `/stream/list` in Section 3.2.1.4 and Table 3-7.

`stream_remove`(*path*, *start=None*, *end=None*) → int

Remove data rows from the stream *path* in the time interval $[start \rightarrow end\rangle$. The stream's internal intervals will be truncated or split as necessary. Returns the total number of data rows that were removed.

If a lot of data is present, removal may take a while. Note also that removal should be done in large or contiguous chunks to maintain overall database performance, as described in the Performance section of the `/stream/remove` documentation in Section 3.2.1.12.

`stream_rename`(*oldpath*, *newpath*)

Rename or relocate a stream in the database from *oldpath* to *newpath*. Metadata and intervals, if any, are relocate to the new path name.

Note that while metadata is moved, the contents of said metadata are unchanged. Any software tools that store and use paths names stored in metadata keys or values will need to update them accordingly.

stream_set_metadata(*path*, *data*)

Set stream metadata for *path*. All existing metadata is replaced with the keys and values in the dictionary *data*. Keys and values must be Unicode text strings.

To insert new keys without affecting existing metadata, use **stream_update_metadata** instead.

stream_update_metadata(*path*, *data*)

Update stream metadata for *path*. The keys and values in dictionary *data* are used to augment or replace the existing metadata in the stream. This is equivalent to:

```
meta = client.stream_get_metadata(path)
meta.update(data)
client.stream_set_metadata(meta)
```

version() → string

Return the version of the NilmDB server as a string, such as `"1.9.7"`.

class nilmdb.client.client.**StreamInserter**(*client*, *path*, *start*, *end*)

An instance of this class returned is returned by **Client().stream_insert_context()** and used to manage the insertion of rows of data into NilmDB. Methods are provided that allow insertion of data, as well as control over the start and end times of the newly created intervals.

Conceptually, this class manages filling of contiguous time intervals on the server, with no gaps, that extend from timestamp *start* to timestamp *end*. Data being inserted has timestamps t that satisfy $start \leq t < end$. If *start* or *end* was not provided, it is deduced from the data timestamps based on the guidelines that follow.

The basic data flow is as follows:

1. The first inserted line begins a new interval that starts at *start*. If *start* was not provided, it is set to the timestamp of the first inserted data row.
2. Subsequent lines go into the same contiguous interval. As lines are inserted, this class may internally make multiple insertion requests to the NilmDB server, but will structure the timestamps to leave no gaps.
3. The current contiguous interval can be ended by manually calling **finalize()**, which the context manager will also do automatically when it completes. This will send any remaining data to the server, using the *end* timestamp to end the final interval. If *end* was not provided, it is set to the last timestamp seen in the data, plus a small delta.[3]

After a **finalize()**, the manager returns to waiting for a new first line to be inserted at step 1, which would start a new interval.

Prior to step 1, **update_start()** can be used to change the start time of the interval that will be created. Similarly, **update_end()** can be used before step 3 to change the end time for the interval.

A basic example of using this class is:

```
with client.stream_insert_context("/test/foo") as ctx:
    ctx.insert("1234567001.0 1 2 3 4\n")
    ctx.insert("1234567002.0 4 3 2 1\n1234567003.0 5 5 5 4\n")
    ctx.update_end(1234567004)
    ctx.finalize()
```

The methods are described below.

finalize()

Stop filling the current contiguous interval. All outstanding data will be sent to the server. If more data is inserted after a call to this function,

[3]A delta of 1 timestamp unit is added, equal to one microsecond given the usual interpretation. This is required because intervals are half-open, which means that they must end *after* the last timestamp that they contain.

it will become part of a new interval, as if a new context manager object were created.

insert(*data*)

Insert a chunk of ASCII formatted data in string form. The passed *data* does not necessarily need to be contain complete lines, but the concatenation of all of the chunks passed to this function should consist of complete lines of text, terminated by newlines.

The text format of data rows is described in the `/stream/insert` documentation on page 108.

send()

Send any data that may have been locally buffered up and not yet sent to the server. This does not affect any other treatment of timestamps, endpoints, or data. In general, this function does not need to be used, but may be helpful to ensure data appears at the server in a timely manner when it is being generated or inserted slowly.

update_end(end)

Update the end time for the current contiguous interval. Call this before **finalize()**.

update_start(start)

Update the start time for the next contiguous interval. Call this before starting to insert data for a new interval, for example, after **finalize()**.

3.2.2.2 Module `nilmdb.client.numpyclient`

This module contains the **NumpyClient** class, which is a subclass of the main **Client** class. It provides additional methods for inserting and extracting data as NumPy arrays.

class `nilmdb.client.numpyclient.`**`NumpyClient`**(*url*)

Base class: `nilmdb.client.client.Client`

Subclass of the main **`Client`** that adds methods for inserting and extracting data as NumPy arrays. This class also provides all of the functions of **`Client`**, and can replace it in all cases; it is provided as a separate class to avoid a dependency on NumPy for situations where NumPy functionality it is not needed.

This class can also be used as as a context manager, using the following structure:

```
with NumpyClient("http://localhost/nilmdb") as client:
    print client.version()
```

When used in this manner, the client, and any connection to the server, will be automatically closed when the `with` block exits, as if `client.close()` had been called.

The new methods introduced by this subclass are described below.

`stream_extract_numpy`(*path*, *start=None*, *end=None*, *layout=None*, *maxrows=100000*, *structured=False*) → *generator*

Extract rows of data from *path* in the interval [*start* → *end*⟩. This function returns a generator that yields NumPy arrays containing the extracted data. Each NumPy array contains up to *maxrows* rows, or fewer if less data is available.

This function needs to know the stream layout in order to create the NumPy array. If *layout* is *None*, it is read using **`stream_info()`**. Manually specifying a layout string in the form `<type>_<count>` avoids this extra request.

The format of the NumPy array depends on the *structured* argument. If *structured* is *False*, data is returned as a homogeneous 2-dimensional array containing `<count>`+1 columns. The first is the timestamp, and the remaining columns are the `<count>` data values. Every element in this

array is of the same NumPy type, either `int64` or `float64`, depending on which type best represents the values in the original stream layout.

If *structured* is *True*, data is returned as a NumPy "structured array" [50], which maintains separate and layout-accurate types for the timestamp and data values. This array is 1-dimensional, and each element contains data for one row. Specifically, given the array `x`, the timestamp of row n is `x[n]["timestamp"]`, and the data for the same row is an array of `<count>` elements `x[n]["data"]`.

Generally, *structured* should be left as *False*, as this makes processing easier. In some cases, this may lose precision because the data (of type `<type>`) and timestamp (of type `int64`) have to be coerced to the same type. Problems arise in two specific cases:

- Layout `<type>` is `uint64`, and timestamp or data values are outside the range $-2^{53} \to 2^{53}$.
- Layout `<type>` is `float32` or `float64`, and timestamp values are outside the range $-2^{53} \to 2^{53}$.

In those situations, the resulting type of the unstructured array, `float64`, is unable to represent the original values exactly. Using a structured array, by setting *structured* to *True*, avoids this issue.

`stream_insert_numpy_context`(*path*, *start=None*, *end=None*, *layout=None*) → **`StreamInserterNumpy`**

Insert data from NumPy arrays into a stream. Returns a **`StreamInserterNumpy`** context manager that allows data to be efficiently inserted into a stream in a piecewise manner. This is the primary function for inserting NumPy array data into NilmDB, and is the NumPy counterpart to the text-based **`stream_insert_context()`** function.

Data is provided to the context manager as NumPy arrays. The arrays are either 2-dimensional with a single data type, or 1-dimensional structured

arrays, as described in the documentation of **stream_extract_numpy()**. The format of the data must match the database layout for the given *path*.

The context manager needs to know the stream layout in order to interpret the NumPy arrays. If *layout* is *None*, it is read using **stream_info()**. Manually specifying a layout string in the form `<type>_<count>` avoids this extra request.

See the **StreamInserterNumpy** documentation below for details on the context manager methods. A basic example of usage is:

```
with client.stream_insert_numpy_context("/test/foo") as ctx:
    ctx.insert(numpy.array([[1234567000.0, 1, 2, 3, 4]]))
    ctx.insert(numpy.array([[1234567001.0, 4, 3, 2, 1]]))
```

stream_insert_numpy(*path*, *data*, *start=None*, *end=None*, *layout=None*)

Insert data from a NumPy array into the stream at *path*. Use the function **stream_insert_numpy_context()** instead; this is a compatibility function that is essentially the same as:

```
with client.stream_insert_numpy_context(path, start,
                                        end, layout) as ctx:
    ctx.insert(data)
```

class nilmdb.client.numpyclient.**StreamInserterNumpy**(*client*, *path*, *start*, *end*, *dtype*)

Base class: nilmdb.client.client.StreamInserter

An instance of this class returned is returned by **NumpyClient().stream_insert_numpy_context()** and used to manage the insertion of rows of data into NilmDB. This function performs exactly like its base class, except that the **insert()** function is replaced with one that takes NumPy arrays rather than text. The data flow and other behaviors are unchanged, and other methods such as `finalize()` are still available.

See the **StreamInserter** documentation on page 125 for more details.

insert(*array*)

Insert rows of data from *array* into the database. *array* can contain any number of rows of data. The format of *array* must match the one of the two formats returned by **stream_extract_numpy()**, documented on page 128. That is, both 1-dimensional (structured) and 2-dimensional (unstructured) arrays are supported. Structured arrays must specifically include the named fields "timestamp" and "data" that match the database types. Unstructured arrays simply contain `<count>`+1 columns, where the first column is the timestamp and the remaining `<count>` columns are the data values for the given row.

3.2.2.3 Module `nilmdb.client.errors`

This module contains the NilmDB-specific exceptions that are raised when errors occur.

exception `nilmdb.client.errors.`**Error**(*status, message, url, traceback*)

Base exception: `Exception`

This is the base of the NilmDB exceptions. This exception is raised when errors are encountered during communication with the NilmDB server. The typical way to catch one of these errors is:

```
try:
    # do something with a Client or NumpyClient
except nilmdb.client.errors.Error as e:
    # use e.status, e.message, e.url, or e.traceback for details;
    # str(e) or repr(e) for complete error string
```

The following member variables are available:

status

A string with the HTTP status of the server response. For example, "400 Bad Request", "404 Not Found", or "404".

message

The specific message returned by the server. Generally, when communicating with a properly functioning NilmDB server, this will contain more detail about the cause of the error.

url

The complete URL that the client library was trying to access.

traceback

A formatted Python traceback indicating the error on the server, if one was sent. By default, the NilmDB server does not send tracebacks for expected user errors such as bad parameters, but will for unexpected errors that may indicate a bug on the server. This is intended for debugging.

exception `nilmdb.client.errors.`**`ClientError`**`(`*status, message, url, traceback*`)`

Base exception: `nilmdb.client.errors.Error`

Subclass of **`Error`** that is raised when **status** is between 400 and 499, indicating that the error likely occurred due to bad parameters, or bad data from the client.

exception `nilmdb.client.errors.`**`ServerError`**`(`*status, message, url, traceback*`)`

Base exception: `nilmdb.client.errors.Error`

Subclass of **`Error`** that is raised when **status** is between 500 and 599, indicating that the error was likely a server problem.

3.2.2.4 Module `nilmdb.utils.time`

This module contains utility functions for manipulating NilmDB timestamps, which are represented as integer microseconds since epoch (1970/01/01 00:00 UTC).

`nilmdb.utils.time.`**`now`**`()` → *int*

Return the current timestamp as integer microseconds since epoch.

String	Meaning
`now`	Current date and time
`min`	$-\infty$ (earliest representable time)
`max`	$+\infty$ (latest representable time)
`1234567890`	Unix timestamp[1] (seconds since epoch)
`1234567890111222`	NilmDB timestamp[1] (microseconds since epoch)
`@1234567890`	NilmDB timestamp[2] (microseconds since epoch)
`@1234567890111222`	NilmDB timestamp[2] (microseconds since epoch)
`Thu, 25 Sep 2003 10:49:41 -0500`	
`Thu Sep 25 10:36:28.51 EST 2003`	
`2003-09-25T10:49:41.51234-05:00`	
`20030925-104941`	
`Sep 25 2003`	
`09/25/2003 10:49:41`	

[1]These cases are distinguished based on magnitude.
[2]The leading `@` forces it to be interpreted as a raw NilmDB timestamp.

Table 3-19: Example formats handled by the NilmDB date and time parser.

`nilmdb.utils.time.`**`parse_time`**`(`*toparse*`)` $\rightarrow$ *int*

Parse the free-form text string *toparse* as a date and time, and return a timestamp as integer microseconds since epoch. The parser handles a wide variety of human-readable date and time formats, as well as NilmDB-specific shortcuts. Table 3-19 shows some examples. Many variations on the human-readable formats are also accepted.

If the interpretation depends on the timezone, and no timezone was included in the string, the local system's `TZ` environment variable is used.

If *toparse* cannot be parsed, `ValueError` is raised.

`nilmdb.utils.time.`**`timestamp_to_human`**`(`*timestamp*`)` $\rightarrow$ *str*

Convert a NilmDB timestamp to a human-readable string, using the system's local timezone (for example, from the `TZ` environment variable). The output format is "Thu, 25 Sep 2003 10:49:41.000000 -0500". If the timestamp represents $-\infty$ or $+\infty$, the output is "(minimum)" or "(maximum)". Intended for display purposes.

`nilmdb.utils.time.`**`rate_to_period`**`(`*hz*, *cycles=1*`)` → *int*

Convert a rate (in Hz) to a period (in NilmDB timestamp units, i.e., integer microseconds). Computes *cycles*/*hz* with unit conversion and rounding.

`nilmdb.utils.time.`**`unix_to_timestamp`**`(`*unix*`)` → *int*

Alias: `nilmdb.utils.time.seconds_to_timestamp()`

Converts the Unix timestamp *unix* (floating point seconds since epoch) to a NilmDB timestamp (integer microseconds since epoch).

`nilmdb.utils.time.`**`timestamp_to_unix`**`(`*timestamp*`)` → *float*

Alias: `nilmdb.utils.time.timestamp_to_seconds()`

Converts the NilmDB timestamp *timestamp* (integer microseconds since epoch) to a Unix timestamp (floating point seconds since epoch).

3.2.2.5 Module `nilmdb.utils.interval`

This module contains classes and utility functions for manipulating NilmDB intervals. Intervals are half-open; they include the starting time but not the end time.

exception `nilmdb.utils.interval.`**`IntervalError`**`()`

Base exception: `Exception`

Exception raised due to interval errors (end preceds start, etc).

class `nilmdb.utils.interval.`**`Interval`**`(`*start*, *end*`)`

Class representing the time interval $[start \rightarrow end\rangle$.

`human_string``()` → str

Return a human-readable representation of this interval, intended for display purposes.

`intersects``(`*other*`)` → bool

Return *True* if this interval intersects *other*.

subset(*start*,*end*) → **Interval**

Return a new **Interval** that is a subset of this one. Raises **IntervalError** if the specified *start* and *end* do not represent a subset of this interval.

nilmdb.utils.interval.**optimize**(*it*) → *generator*

Given an iterable or list *it* of **Interval** objects, optimize them by joining together intervals that are adjacent in time, and return a generator that yields the new intervals. For example, intervals

$$[1 \to 2\rangle\,, [2 \to 3\rangle\,, [4 \to 5\rangle\,, [5 \to 6\rangle$$

would be optimized to:

$$[1 \to 3\rangle\,, [4 \to 6\rangle$$

nilmdb.utils.interval.**set_difference**(*a*, *b*) → *generator*

Compute the set-difference $a \setminus b$ between the intervals in a and the intervals in b; that is, the ranges that are present in a but not present in b.

a and b must both be iterables containing **Interval** objects. Returns a generator that yields each output interval in turn. Output intervals are built as subsets of the intervals in a.

nilmdb.utils.interval.**intersection**(*a*, *b*) → *generator*

Compute the intersection $a \cap b$ between the intervals in a and the intervals in b; that is, the ranges that are present in both a and b.

a and b must both be iterables containing **Interval** objects. Returns a generator that yields each output interval in turn. Output intervals are built as subsets of the intervals in a.

3.2.3 Command Line Reference

The installation of NilmDB includes command-line programs that provide a user interface to database and data management tasks. The main component is `nilmtool`, a monolithic multi-purpose program that provides many subcommands. The overall operation of `nilmtool`, as well as the operation of each specific subcommand, are detailed independently in the following sections.

The utility program `nilmdb-server`, which executes a standalone copy of the NilmDB server, is described in Section 3.2.3.13. Finally, the `nilmdb-fsck` program, designed to verify database consistency and fix problems that can arise from corruption or improper shutdowns, is described in Section 3.2.3.14.

Command-line arguments can often be supplied in both short and long forms, and many arguments are optional. The following documentation uses these conventions:

- An argument that takes an additional parameter is denoted `-f` *`FILE`*.
- The syntax `-f` *`FILE`*`, --file` *`FILE`* indicates that either the short form (`-f`) or long form (`--file`) can be used interchangeably.
- Square brackets (`[]`) denote optional arguments.
- Pipes (`A | B`) indicate that either `A` or `B` can be specified, but not both.
- Curly braces (`{}`) indicate a list of mutually-exclusive argument choices.

Many of the programs support arguments that represent a NilmDB timestamp. This timestamp is specified as a free-form string, as supported by the `parse_time` client library function, described in Section 3.2.2.4. Examples of accepted formats are shown in Table 3-19 on page 133.

3.2.3.1 `nilmtool` – Multipurpose NilmDB management tool

Usage

```
nilmtool [-h] [-v] [-u URL] {help, info, create, rename, list, intervals,
         metadata, insert, extract, remove, destroy} ...
```

Description

Multipurpose tool that provides command-line access to most of the NilmDB functionality. The command-line syntax provides the ability to execute *subcommands*: first, global arguments that affect the behavior of all subcommands can be specified, followed by one subcommand name, followed by arguments for that subcommand. Each defines its own arguments and is documented independently.

Help and Version

`-h, --help`

Print a help message with usage information and details on all supported command-line arguments. This can also be specified after the subcommand, in which case the usage and arguments of the subcommand are shown instead.

`-v, --version`

Print the `nilmtool` version.

Arguments

`-u` *`URL`*`, --url` *`URL`* *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL. Must be specified before the subcommand.

`subcommand ...`

The subcommand to run, followed by its arguments. This is required.

Environment Variables

Some behaviors of `nilmtool` subcommands can be configured via environment variables.

`NILMDB_URL` *(default:* `http://localhost/nilmdb/`*)*

The default URL of the NilmDB server. This is used if `--url` is not specified, and can be set as an environment variable to avoid the need to specify it on each invocation of `nilmtool`.

`TZ` *(default:* system default timezone*)*

The timezone to use when parsing or displaying times. This is usually of the form `America/New_York`, using the standard TZ names from the IANA Time Zone Database [51].

3.2.3.2 `nilmtool help` – Print help for a subcommand

Usage

```
nilmtool help [-h] subcommand
```

Description

Print more specific help for a subcommand. `nilmtool help subcommand` is the same as `nilmtool subcommand --help`.

3.2.3.3 `nilmtool info` – Server information

Usage

```
nilmtool info [-h]
```

Description

Print server information such as software versions, database location, and disk space usage.

Examples

```
$ nilmtool info
Client version: 1.9.7
Server version: 1.9.7
Server URL: http://localhost/nilmdb/
Server database path: /home/nilmdb/db
Server disk space used by NilmDB: 143.87 GiB
Server disk space used by other: 378.93 GiB
Server disk space reserved: 6.86 GiB
Server disk space free: 147.17 GiB
```

3.2.3.4 `nilmtool create` – Create a new stream

Usage

`nilmtool create` [-h] *PATH LAYOUT*

Description

Create a new empty stream at the specified path and with the specified layout.

Required Arguments

PATH

Path of the new stream. Stream paths are similar to filesystem paths and must contain at least two components. For example, `/foo/bar`.

LAYOUT

Layout for the new stream. Layouts are of the form `<type>_<count>`. The `<type>` is one of those described in Section 2.2.3, such as `uint16`, `int64`, or `float32`. `<count>` is a numeric count of how many data elements there are, per row. Streams store rows of homogeneous data only, and the largest supported `<count>` is 1024. Generally, counts should fall within a much lower range, typically between 1 and 32. For example, `float32_8`.

3.2.3.5 `nilmtool rename` – Rename a stream

Usage

```
nilmtool rename [-h] OLDPATH NEWPATH
```

Description

Rename or relocate a stream in the database from one path to another. Metadata and intervals, if any, are relocated to the new path name.

Required Arguments

OLDPATH

Old existing stream path, e.g. `/foo/old`

NEWPATH

New stream path, e.g. `/foo/bar/new`

Notes

Metadata *contents* are not changed by this operation. Any software tools that store and use path names stored in metadata keys or values will need to update them accordingly.

3.2.3.6 `nilmtool list` – List streams

Usage

```
nilmtool list [-h] [-E] [-d] [-s TIME] [-e TIME] [-T] [-l] [-n]
              [PATH [PATH ...]]
```

Description

List streams available in the database, optionally filtering by path, and optionally including extended stream info and intervals.

Arguments

PATH *(default: *)*

If paths are specified, only streams that match the given paths are shown. Wildcards are accepted; for example, `/sharon/*` will list all streams with a path beginning with `/sharon/`. Note that, to prevent wildcards from being interpreted by the shell, they should be quoted at the command line; for example:

```
nilmtool list "/sharon/*"
nilmtool list "*raw"
```

Output Options

`-E, --ext`

Show extended stream information, like interval extents, total rows of data present, and total amount of time covered by the stream's intervals.

`-T, --timestamp-raw`

When displaying timestamps in the output, show raw timestamp values from the NilmDB database rather than converting to human-readable times. Raw values are typically measured in microseconds since the Unix time epoch (1970/01/01 00:00 UTC).

`-l, --layout`

Display the stream layout next to the path name.

`-n, --no-decim`

Omit streams with paths containing the string "`~decim-`", to avoid cluttering the output with decimated streams.

Interval Detail

`-d, --detail`

In addition to the normal output, show the time intervals present in each stream. See also `nilmtool intervals` in Section 3.2.3.7, which can display more details about the intervals.

`-s` *`TIME`*`, --start` *`TIME`* *(default:* `min`*)*

Starting timestamp for intervals (free-form, inclusive).

`-e` *`TIME`*`, --end` *`TIME`* *(default:* `max`*)*

Ending timestamp for intervals (free-form, noninclusive).

3.2.3.7 `nilmtool intervals` – List intervals

Usage

```
nilmtool intervals [-h] [-d DIFFPATH] [-s TIME] [-e TIME] [-T] [-o]
                   PATH
```

Description

List intervals in a stream, similar to `nilmtool list --detail` *`path`*, but with options for calculating set-differences between intervals of two streams, and for optimizing the output by joining adjacent intervals.

Arguments

`PATH`

List intervals for this path.

`-d` *`DIFFPATH`*`, --diff` *`DIFFPATH`* *(default:* none*)*

If specified, perform a set-difference by subtract the intervals in this path; that is, only show interval ranges that are present in the original *`path`* but not present in *`diffpath`*.

`-s TIME, --start TIME` *(default:* `min`)

Starting timestamp for intervals (free-form, inclusive).

`-e TIME, --end TIME` *(default:* `max`)

Ending timestamp for intervals (free-form, noninclusive).

Output Options

`-T, --timestamp-raw`

When displaying timestamps in the output, show raw timestamp values from the NilmDB database rather than converting to human-readable times. Raw values are typically measured in microseconds since the Unix time epoch (1970/01/01 00:00 UTC).

`-o, --optimize`

Optimize the interval output by merging adjacent intervals. For example, the two intervals $[1 \to 2\rangle$ and $[2 \to 5\rangle$ would be displayed as one interval $[1 \to 5\rangle$.

3.2.3.8 `nilmtool metadata` – Manage stream metadata

Usage

```
nilmtool metadata [-h] PATH [-g [KEY ...] | -s KEY=VALUE [...] |
                            -u KEY=VALUE [...] | -d [KEY ...]]
```

Description

Get, set, update, or delete the key/value metadata associated with a stream.

Arguments

path

Path of the stream for which to manage metadata. Required, and must be specified before the action arguments.

Action Arguments

These actions are mutually exclusive.

`-g [KEY ...], --get [KEY ...]` *(default:* all)

Get and print metadata for the specified key(s). If none are specified, print metadata for all keys. Keys are printed as `key=value`, one per line.

`-s [KEY=VALUE ...], --set [KEY=VALUE ...]`

Set metadata. Keys and values are specified as a `key=value` string. This replaces *all* existing metadata on the stream with the provided keys; any keys present in the database but not specified on the command line are removed.

`-u [KEY=VALUE ...], --update [KEY=VALUE ...]`

Update metadata. Keys and values are specified as a `key=value` string. This is similar to `--set`, but only adds or changes metadata keys; keys that are present in the database but not specified on the command line are left unchanged.

`-d [KEY ...], --delete [KEY ...]` *(default:* all)

Delete metadata for the specified key(s). If none are specified, delete *all* metadata for the stream.

Example

```
$ nilmtool metadata /temp/raw --set "location=Honolulu, HI" "source=NOAA"
$ nilmtool metadata /temp/raw --get
location=Honolulu, HI
source=NOAA
$ nilmtool metadata /temp/raw --update "units=F"
location=Honolulu, HI
source=NOAA
units=F
```

3.2.3.9 `nilmtool insert` – Insert data

Usage

```
nilmtool insert [-h] [-q] [-t] [-r RATE] [-s TIME | -f] [-e TIME]
                PATH [FILE]
```

Description

Insert data into a stream. This is a relatively low-level interface analogous to the `/stream/insert` HTTP interface described in Section 3.2.1.13. This is the program that should be used when a fixed quantity of text-based data is being inserted into a single interval, with a known start and end time. If the input data does not already have timestamps, they can be optionally added based on the start time and a known data rate.

In many cases, using the separate `nilm-insert` program is preferable, particularly when dealing with large amounts of pre-recorded data, or when streaming data from a live source. `nilm-insert` is described in Section 3.4.2.6.

Arguments

PATH

Path of the stream into which to insert data. The format of the input data must match the layout of the stream.

FILE *(default:* standard input)

Input data filename, which must be formatted as uncompressed plain text. Default is to read the input from `stdin`.

`-q, --quiet`

Suppress printing unnecessary messages.

Timestamping

To add timestamps to data that does not already have it, specify both of these arguments. The added timestamps are based on the interval start time and the given data rate.

`-t, --timestamp`

Add timestamps to each line

`-r` *`RATE`*`, --rate` *`RATE`*

Data rate, in Hz

Start Time

The start time may be manually specified, or it can be determined from the input filename, based on the following options.

`-s` *`TIME`*`, --start` *`TIME`*

Starting timestamp for the new interval (free-form, inclusive)

`-f, --filename`

Use filename to determine start time

End Time

The ending time should be manually specified. If timestamps are being added, this can be omitted, in which case the end of the interval is set to the last timestamp plus one microsecond.

`-e` *`TIME`*`, --end` *`TIME`*

Ending timestamp for the new interval (free-form, noninclusive)

3.2.3.10 `nilmtool extract` – Extract data

Usage

```
nilmtool insert [-h] -s TIME -e TIME [-B] [-b] [-a] [-m] [-T] [-c]
                PATH
```

Description

Extract rows of data from a specified time interval in a stream, or output a count of how many rows are present in the interval.

Arguments

PATH

Path of the stream from which to extract data.

`-s` *TIME*, `--start` *TIME*

Starting timestamp to extract (free-form, inclusive)

`-e` *TIME*, `--end` *TIME*

Ending timestamp to extract (free-form, noninclusive)

Output Formatting

`-B, --binary`

Output raw binary data instead of the usual text format. For details on the text and binary formatting, see the documentation of HTTP call `/stream/insert` in Section 3.2.1.13.

`-b, --bare`

Omit timestamps from each line of the output.

`-a, --annotate`

Include comments at the beginning of the output with information about the stream. Comments are lines beginning with `#`.

`-m, --markup`

Include comments in the output with information that denotes where the stream's internal intervals begin and end. See the documentation of the `markup` parameter to HTTP call `/stream/extract` in Section 3.2.1.14 for details on the format of the comments.

`-T, --timestamp-raw`

Use raw integer timestamps in the `--annotate` output instead of human-readable strings.

`-c, --count`

Instead of outputting the data, output a count of how many rows are present in the given time interval. This is fast as it does not transfer the data from the server.

3.2.3.11 `nilmtool remove` – Remove rows of data

Usage

```
nilmtool remove [-h] -s TIME -e TIME [-q] [-c] PATH [PATH ...]
```

Description

Remove all data from a specified time range within the stream at /PATH/. Multiple streams may be specified, and wildcards are supported; the same time range will be removed from all matching streams.

Arguments

PATH

Path(s) of streams. Wildcards are supported. At least one path must provided.

`-s TIME, --start TIME`

Starting timestamp of data to remove (free-form, inclusive, required).

`-e TIME, --end TIME`

Ending timestamp of data to remove (free-form, noninclusive, required).

Output Format

`-q, --quiet`

By default, matching path names are printed when removing from multiple paths. With this option, path names are not printed.

`-c, --count`

Display a count of the number of rows of data that were removed from each path.

Example

```
$ nilmtool remove -s @1364140671600000 -e @1364141576585000 -c "/sh/raw*"
Removing from /sh/raw
7239364
Removing from /sh/raw~decim-4
1809841
Removing from /sh/raw~decim-16
452460
```

3.2.3.12 `nilmtool destroy` – Destroy a stream

Usage

`nilmtool destroy [-h] [-R] [-q] PATH [PATH ...]`

Description

Destroy the stream at the specified path(s); the opposite of `nilmtool create`. Metadata related to the stream is permanently deleted. All data must be removed before a stream can be destroyed. Wildcards are supported.

Arguments

PATH

Path(s) of streams. Wildcards are supported. At least one path must provided.

`-R, --remove`

If specified, all data is removed before destroying the stream. Equivalent to first running `nilmtool remove -s min -e max` *path*.

`-q, --quiet`

Don't display names when destroying multiple paths

3.2.3.13 `nilmdb-server` – Standalone NilmDB server

Usage

```
nilmdb-server [-h] [-v] [-a ADDRESS] [-p PORT] [-d DATABASE]
              [-q] [-t] [-y]
```

Description

Run the standalone NilmDB server. Note that the NilmDB server is typically run as a WSGI process as described in Section 3.1.1.3. This program runs NilmDB using a built-in web server instead.

Arguments

`-v, --version`

Print the installed NilmDB version.

`-a` *ADDRESS*`, --address` *ADDRESS* *(default:* `0.0.0.0`)

Only listen on the given IP address. The default is to listen on all addresses.

`-p PORT, --port PORT` *(default:* `12380`*)*

Listen on the given TCP port.

`-d DATABASE, --database DATABASE` *(default:* `./db`*)*

Local filesystem directory of the NilmDB database.

`-q, --quiet`

Silence output.

Debug Options

`-t, --traceback`

Provide tracebacks in the error response for client errors (HTTP status codes 400 - 499). Normally, tracebacks are only provided for server errors (HTTP status codes 500 - 599).

`-y, --yappi`

Run under the yappi profiler and invoke an interactive shell afterwards. Not intended for normal operation.

3.2.3.14 `nilmdb-fsck` – Database check and repair

Usage

`nilmdb-fsck [-h] [-v] [-f] [-n] DATABASE`

Description

Check database consistency, and optionally repair errors automatically, when possible. Running this may be necessary after an improper shutdown or other corruption has occurred. This program will refuse to run if the database is currently locked by any other process, like the Apache webserver; such programs should be stopped first.

Help and Version

`-h, --help`

Print a help message with usage information and details.

`-v, --version`

Print the installed NilmDB version. Generally, you should ensure that the version of `nilmdb-fsck` is newer than the NilmDB version that created, or last used, the given database.

Arguments

`DATABASE`

Local filesystem directory of the NilmDB database to check.

`-f, --fix`

Attempt to fix errors when possible. Note that this may involve removing intervals or data.

`-n, --no-data`

Skip the slow full-data check. The earlier, faster checks are likely to find most database corruption, so the data checks may be unnecessary.

3.3 NilmRun Interfaces

NilmRun is a server that facilities the execution of arbitrary programs and code on a remote server, via HTTP. An overview can be found in Section 2.4.2.

3.3.1 HTTP API

The NilmRun HTTP API follows the same structure as the NilmDB HTTP API, which is described in Section 3.2.1. Details on request methods, query parameter formats, response types, and error bodies can be found there.

The API consists of a series of resources that are accessed by Uniform Resource Identifier (URI). For example, the resource for retrieving a list of managed processes is `/process/list`. The base URL of the NilmRun installation is prefixed to this URI to build a complete request URL, such as `http://<host>/nilmrun/process/list`.

In the example API transactions that follow, requests to the server are denoted with a light blue line in the margin, and responses are denoted with a dark green line. Whitespace may be added to JSON responses for readability. Only relevant headers are included in both the request and response.

3.3.1.1 `/` – NilmRun short status text

Synopsis

```
GET /
```

Description

Returns a human-readable string with the NilmRun version number and local hostname. This is intended to assist with configuration and testing during installation, and is the default page that will be returned if a user visits the NilmRun base URL with a web browser.

Request Parameters

None

Response

Returns a `text/plain` string of human-readable text.

Example

```
GET / HTTP/1.1

HTTP/1.1 200 OK
Content-type: text/plain;charset=utf-8

This is NilmRun version 1.3.3, running on host pilot.lees.
```

3.3.1.2 `/version` – Get NilmRun server version

Synopsis

```
GET /version
```

Description

Returns the NilmRun server version number in JSON format.

Request Parameters

None

Response

Returns a JSON object containing a single string. The version number follows typical software version number conventions (i. e., 1.7.10 is newer than 1.7.9).

Example

```
GET /version HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

"1.3.3"
```

3.3.1.3 `/run/command` – Execute a command on the server

Synopsis

```
POST /run/command
{ "argv": [ "nilm-decimate-auto", "/foo/bar" ] }
POST /run/command
{ "argv": [ "sh", "-c", "nilm-decimate-auto /foo/bar" ] }
```

Description

Execute an arbitrary program on the server with the given arguments. NilmRun will spawn a new process and execute the command as given, returning an

identifier that can be used to refer to the process and its children later. Output and error codes from the command are collected, and can be retrieved using the `/process/status` request (Section 3.3.1.6). The process and its children must be later removed from the manager, using `/process/remove` (Section 3.3.1.7).

Request Parameters

Parameters are shown in Table 3-20. Required parameters are marked with ∗. The request body should be of type `application/json`.

∗	Name	Description
∗	`argv`	List of strings with the command and arguments. The list must have at least one entry, the command. Subsequent entries are passed as arguments to that command.

Table 3-20: Parameters for the `/run/command` HTTP request.

The first element of the arguments, `argv[0]`, is the command. If the command is not on the remote system's `PATH`, a full path should be specified.

The command is executed directly, not through the shell. If shell features are needed, such as pipelines, redirection, wildcards, or variable expansion, the shell can be explicitly invoked as `sh -c`, with the full shell command line as the third entry in the `argv` list. For example:

```
{ "argv": [ "sh", "-c", "ls $HOME/ | sort" ] }
```

When using the shell, spaces and special characters must be escaped and quoted accordingly.

Response

As soon as the process is started, `200 OK` is returned, and the body is a JSON object containing a unique process identifier (pid) for the executed process. The pid is a string of characters, typically in UUID Version 1 format [52]. It is an opaque token that is used to refer to the executed process and its children later.

Errors

If the arguments are formatted incorrectly, or an early error occurs while executing the command, `400 Bad Request` is returned, and the `message` field in the response body will contain more information about the error. Note that errors in executing the command may not be detected immediately, particularly when running a command through the shell. In that case, the response will initially indicate success, and errors can be found in later requests for process status (Section 3.3.1.6).

Example

```
POST /run/command HTTP/1.1
Content-type: application/json

{ "argv": [ "nilm-decimate-auto", "/foo/bar" ] }
HTTP/1.1 200 OK
Content-type: application/json

"d9dedff8-f242-11e2-a79b-000000004e8d"
```

```
POST /run/command HTTP/1.1
Content-type: application/json

{ "argv": [ "/no-such-command" ] }
HTTP/1.1 400 Bad Request
Content-type: application/json

{ "status": "400 Bad Request",
  "message": "[Errno 2] No such file or directory",
  "traceback": ""
}
```

3.3.1.4 `/run/code` – Execute Python code on the server

Synopsis

```
POST /run/code
{ "code": "for x in range(10):\n    print x\n",
  "args": [ ] }
```

Description

Execute arbitrary Python code on the server with the given arguments. The code, which is provided as a string, is executed as if the string were written into a temporary file, and that file were executed by the same Python interpreter as the NilmRun server.

As with `/run/command` (Section 3.3.1.3), output and error codes from the Python script are collected and can be retrieved using the `/process/status` request (Section 3.3.1.6). The process and its children must be later removed from the manager using `/process/remove` (Section 3.3.1.7).

Request Parameters

Parameters are shown in Table 3-21. Required parameters are marked with ∗. The request body should be of type `application/json`.

∗	Name	Description
∗	`code`	String containing formatted Python code to be executed.
	`args`	List of strings to be passed to the Python code as command-line arguments.

Table 3-21: Parameters for the `/run/code` HTTP request.

The code in `code` must be correctly formatted Python code, including embedded newlines and indentation. The arguments in `args` are passed as if they were command-line arguments to a standalone Python script, and will be available to the running code as `sys.argv[1:]`.

Response

As soon as the process is started, `200 OK` is returned, and the body is a JSON object containing a unique process identifier (pid) for the executed process. The pid is a string of characters, typically in UUID Version 1 format [52]. It is an opaque token that is used to refer to the executed process and its children later.

Errors

If the arguments are formatted incorrectly, or an early error occurs while preparing or executing the code, `400 Bad Request` is returned, and the `message` field in the response body will contain more information about the error. Note that some errors, such as parse errors in the Python code may not be detected immediately. In that case, the response will initially indicate success and errors can be found in later requests for process status (Section 3.3.1.6).

Example

```
POST /run/code HTTP/1.1
Content-type: application/json

{ "code": "import sys\nprint 'my args were:', sys.argv[1:]\n"
  "args": [ "hello", "world"] }
HTTP/1.1 200 OK
Content-type: application/json

"e8ffaf5e-f244-11e2-8152-000000005128"
```

```
POST /run/code HTTP/1.1
Content-type: application/json

{ "code": "import nilmtools.copy_one\nnilmtools.copy_one.main()\n",
  "args": [ "/test/foo", "/test/bar" ] }
HTTP/1.1 200 OK
Content-type: application/json

"1ebc4921-f251-11e2-9ee7-000000005b7c"
```

3.3.1.5 `/process/list` – List managed processes

Synopsis

```
GET /process/list
```

Description

Lists the process identifiers (pids) currently being managed by the NilmRun server. This includes processes that are still running as well as processes that have exited but have not yet been removed from the server.

Request Parameters

None

Response

A single JSON list containing pid strings of all processes known to the system. If no processes are currently managed, an empty list is returned. Processes are returned in no particular order.

Example

```
GET /process/list HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

[ "bf700bfa-f25a-11e2-9c75-000000006641",
  "452bb60c-f25a-11e2-bc54-0000000065e2",
  "e8ffaf5e-f244-11e2-8152-000000005128" ]
```

```
GET /process/list HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

[ ]
```

3.3.1.6 `/process/status` – Get process status and output

Synopsis

```
GET /process/status?pid=452bb60c-f25a-11e2-bc54-0000000065e2
GET /process/status?pid=452bb60c-f25a-11e2-bc54-0000000065e2&clear=1
```

Description

Retrieve information about a process managed by the NilmRun server. This includes status information about whether the process is alive, as well as the captured log output from the process.

Request Parameters

Parameters are shown in Table 3-22. Required parameters are marked with $*$.

*	Name	Description
*	`pid`	Process ID string of the process for which to retrieve status.
	`clear`	If specified and nonzero, additionally clear the stored log after returning it.

Table 3-22: Parameters for the `/process/status` HTTP request.

Response

Returns a JSON dictionary containing information for the requested process. The keys of this dictionary and their values are shown in Table 3-23.

Key	Value
`pid`	The process identifier. Matches the pid in the request.
`alive`	`true` if the process is still running, or `false` otherwise.
`exitcode`	An integer specifying the exit code of the process, or `nil` if the process is still running. Zero indicates successful exit.
`start_time`	The time at which this process was started, measured in seconds since Unix epoch (1970/01/01 00:00 UTC).
`log`	Captured contents of the `stdout` and `stderr` output from the process and any children it created, merged and returned as a single string. The output of the command is assumed to have been encoded as UTF-8.

Table 3-23: Contents of the `/process/status` HTTP response.

In the exit code, zero indicates successful exit of the process, and nonzero indicates failure. Generally, positive values indicate that the process exited with an error, while negative values indicate that the process was killed by a signal.

Errors

If `pid` is missing or malformed, `400 Bad Request` is returned. If `pid` does not exist, `404 Not Found` is returned.

Example

```
GET /process/status?pid=452bb60c-f25a-11e2-bc54-0000000065e2 HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

{ "pid": "452bb60c-f25a-11e2-bc54-0000000065e2",
  "alive": true,
  "exitcode": nil,
  "start_time": 1374448029.712343,
  "log": "Source: /test/foo\nCopying data..."
}
```

```
GET /process/status?pid=452bb60c-f25a-11e2-bc54-0000000065e2 HTTP/1.1

HTTP/1.1 200 OK
Content-type: application/json

{ "pid": "452bb60c-f25a-11e2-bc54-0000000065e2",
  "alive": false,
  "exitcode": 0,
  "start_time": 1374448029.712343,
  "log": "Source: /test/foo\nCopying data... done\nCopy complete.\n"
}
```

3.3.1.7 `/process/remove` – Remove a process

Synopsis

```
POST /process/remove
{ "pid": "452bb60c-f25a-11e2-bc54-0000000065e2" }
```

Description

Remove a process that had been started by the NilmRun server. The process is removed from the list of managed processes, and any saved data about the process such as exit status and log output are removed.

This is similar to the Unix concept of "reaping" a process, and must be performed on all processes managed by NilmRun, even if they have already exited. If the process is still running when this request is made, it (and its children) are terminated by sending a `SIGTERM`, followed shortly by a `SIGKILL`.

Request Parameters

Parameters are shown in Table 3-24. Required parameters are marked with ∗.

∗	Name	Description
∗	`pid`	Process ID string of the process to remove.

Table 3-24: Parameters for the `/process/remove` HTTP request.

Response

On success, the response is `200 OK` and the response body is the last available process status before it was removed, in the same format as the `/process/status` request (Section 3.3.1.6).

Note that, if the process is still running, this request may take several seconds to complete while the process is terminated.

Errors

If `pid` is missing or malformed, `400 Bad Request` is returned. If `pid` does not exist, `404 Not Found` is returned. In the rare condition that the process was still running but did not respond to `SIGKILL`, `503 Service Unavailable` may be returned, and the process will remain in the list of managed processes. This is not expected in normal operation.

Example

```
POST /process/remove HTTP/1.1
Content-type: application/json

{ "pid": "e8ffaf5e-f244-11e2-8152-000000005128" }
HTTP/1.1 200 OK
Content-type: application/json

{ "pid": "e8ffaf5e-f244-11e2-8152-000000005128",
  "alive": false,
  "exitcode": 1,
  "start_time": 1374448234.73051,
  "log": "sh: no-such-command: command not found\n"
}
```

3.3.1.8 `/process/info` – Get system performance data

Synopsis

```
GET /process/info
```

Description

Retrieve system-wide information about CPU usage, memory usage, process counts, and I/O transfer rates, and additionally returns the same information on a per-process basis. This information can be used to help measure and distribute the load of various processes across NilmRun servers.

Request Parameters

None

Response

Returns a JSON dictionary which contains nested dictionaries with system information. The format of the top-level dictionary is described in Table 3-25, and the formats of the nested dictionaries follow in Tables 3-26 and 3-27.

Key	Value
`system`	JSON dictionary with system-wide information. The contents of this dictionary are further described in Table 3-26.
`pids`	JSON dictionary with per-process information about every pid managed by NilmRun. Each key is a pid, and the values are *another* dictionary with information about that pid, further described in Table 3-27.
`total`	JSON dictionary with aggregate information about all managed processes. This represents the sum of all of the usage information in the `pids` entry, and is the same format described in Table 3-27.

Table 3-25: Contents of the `/process/info` HTTP response.

Key	Value
`cpu_percent`	Current CPU usage percentage. May be greater than 100% for multi-core systems.
`cpu_max`	Maximum CPU usage percentage if all cores were fully loaded.
`procs`	Total number of running processes, including those not managed by NilmRun.
`mem_used`	Total amount of physical memory used, in bytes.
`mem_total`	Total amount of physical memory available, in bytes.

Table 3-26: Contents of the system-wide `/process/info` dictionary.

Key	Value
`cpu_percent`	Current CPU usage percentage. May be greater than 100% if multiple cores are being used.
`cpu_user`	Amount of CPU time, in seconds, spent executing instructions of the managed process.
`cpu_sys`	Amount of CPU time, in seconds, spent in the system while executing tasks on behalf of the managed process.
`mem_phys`	Amount of physical memory used by the process, in bytes.
`mem_virt`	Virtual memory size of the process, in bytes.
`io_read`	I/O bytes read (from disk or network) by the process.
`io_write`	I/O bytes written (to disk or network) by the process.
`procs`	Number of *running* system-level processes corresponding to this NilmRun pid.

Table 3-27: Contents of the process-level `/process/info` dictionary.

Note that one NilmRun pid can correspond to multiple system-level processes, if the executed command or code created them. As such, the percentages and counts in Table 3-27 are aggregated across all system-level processes.

Measurement or rounding errors may cause some inaccuracies in the results. For example, `cpu_percent` may exceed `cpu_max`. All measurements should be considered to be estimates and interpreted accordingly. If a process has terminated, some data such as `io_read` may be unavailable and returned as zero.

Performance Notes

This request takes at least one second, in order to collect CPU usage statistics.

Example

```
GET /process/info HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-type: application/json

{
    "system": {
        "cpu_percent": 188.7,
        "cpu_max": 200,
        "procs": 245,
        "mem_used": 5651126341,
        "mem_total": 8396918784
    },
    "pids": {
        "16b95423-f277-11e2-8f03-000000000672": {
            "cpu_percent": 4,
            "cpu_user": 1.29,
            "cpu_sys": 3.3,
            "mem_phys": 1265664,
            "mem_virt": 13963264,
            "io_read": 52875264,
            "io_write": 0,
            "procs": 2
        },
        "46e7ea3d-f272-11e2-8dab-000000007f41": {
            "cpu_percent": 101,
            "cpu_user": 2211.02,
            "cpu_sys": 0.26,
            "mem_phys": 4096,
            "mem_virt": 151552,
            "io_read": 0,
            "io_write": 0,
            "procs": 1
        }
    },
    "total": {
        "cpu_percent": 105,
        "cpu_user": 2212.31,
        "cpu_sys": 3.56,
        "mem_phys": 1269760,
        "mem_virt": 14114816,
        "io_read": 52875264,
        "io_write": 0,
        "procs": 3
    }
}
```

3.3.2 Command Line Reference

NilmRun includes a command line program that allows the server to be run in a standalone mode, using a built-in web server. Additionally, a set of tools is offered for running, listing, and removing processes.

The conventions used for the command-line documentation are the same as those described in Section 3.2.3.

3.3.2.1 `nilmrun-server` – Standalone NilmRun server

Usage

```
nilmrun-server [-h] [-v] [-a ADDRESS] [-p PORT] [-q] [-t]
```

Description

Run the standalone NilmRun server. Note that the NilmRun server is typically run as a WSGI process, as described in Section 3.1.2.3. Running it in standalone mode may be insecure, as no access control or authentication is supported.

Arguments

`-v, --version`

Print the installed NilmRun version.

`-a ADDRESS, --address ADDRESS` *(default:* `0.0.0.0`*)*

Only listen on the given IP address. The default is to listen on all addresses.

`-p PORT, --port PORT` *(default:* `12381`*)*

Listen on the given TCP port.

`-q, --quiet`

Silence output.

Debug Options

`-t, --traceback`

Provide tracebacks in the error response for client errors (HTTP status codes 400 - 499). Normally, tracebacks are only provided for server errors (HTTP status codes 500 - 599).

3.3.2.2 `nilmrun-ps` – List processes

Usage

`nilmrun-ps` `[-h] [-v] [-u` *`URL`*`] [-n]`

Description

List processes on a remote NilmRun server. Shows overall system information as well as detailed information about each process.

Arguments

`-u` *`URL`*`, --url` *`URL`* *(default:* `http://localhost/nilmrun/`*)*

NilmRun server URL. For servers that require authentication, it can be included in the URL in the form `http://`*`user`*`:`*`password`*`@`*`host`*`/`.

`-n, --noverify`

Disable SSL certificate verification.

Environment Variables

`NILMRUN_URL` *(default:* `http://localhost/nilmdb/`*)*

The default URL of the NilmRun server.

Output

The output of `nilmrun-ps` includes overall system information about the number of running processes, CPU usage, and memory usage on the server. Process information is shown in a table with the following columns:

`PID`

Process ID.

`STATE`

Process status. This is "alive" if the process is still running, "done" if it has exited successfully, and "error" if it has exited with an error.

`SINCE`

Date and time that the process was started.

`PROC`

Number of operating system processes associated with this NilmRun process.

`CPU`

CPU usage of this process as a percentage of a single CPU core.

`LOG`

Length, in bytes, of the stored output for the process. This output can be retrieved when the process is removed with `nilmrun-kill`.

Example

```
procs: 2 nilm, 157 other
  cpu: 29% nilm, 96% other, 200% max
  mem: 623 MiB used, 3965 MiB total, 16%
PID                                  STATE SINCE          PROC CPU LOG
76d81854-feca-11e2-9b2c-000000002559 alive 08/06-15:00:30 2    3   1337
8168f08f-feca-11e2-87fe-00000000255c alive 08/06-15:00:48 1    26  45056
```

3.3.2.3 `nilmrun-run` – Run a command on a NilmRun server

Usage

nilmrun-run [-h] [-v] [-u *URL*] [-n] [-d] *CMD* [*ARG* [...]]

Description

Run a command on a NilmRun server. By default, this program will poll the command's output log and display it while waiting for the process to exit.

Arguments

`-u` *URL*, `--url` *URL* *(default:* `http://localhost/nilmrun/`*)*

NilmRun server URL. For servers that require authentication, it can be included in the URL in the form `http://`*user*`:`*password*`@`*host*`/`.

`-n, --noverify`

Disable SSL certificate verification.

Remote Program

`-d, --detach`

Run process and return immediately, without printing the command output. The process must be later removed with `nilmrun-kill`.

CMD [*ARG* [...]]

Remote command to execute, with arguments.

Environment Variables

`NILMRUN_URL` *(default:* `http://localhost/nilmdb/`*)*

The default URL of the NilmRun server.

3.3.2.4 `nilmrun-kill` – Kill/remove a process

Usage

```
nilmrun-kill [-h] [-v] [-u URL] [-n] [-q] PID [[...]
```

Description

Kill or remove a process from the NilmRun server. This terminates all system-level processes that are running and removes the entry from the NilmRun process listing. Stored log output of the command, if any, is displayed after the process is removed.

Arguments

`-u URL, --url URL` *(default:* `http://localhost/nilmrun/`*)*

Nilmrun server URL. For servers that require authentication, it can be included in the URL in the form `http://user:password@host/`.

`-n, --noverify`

Disable SSL certificate verification.

Remote Program

`-q, --quiet`

Omit display of the command's final output log.

`PID [...]`

One or more process IDs to remove. Process IDs should be those listed by `nilmrun-ps`.

`NILMRUN_URL` *(default:* `http://localhost/nilmdb/`*)*

The default URL of the Nilmrun server.

3.4 NilmTools Interfaces

NilmTools[4] is a loosely-knit collection of additional Python programs and libraries for manipulating data in the NilmDB database.

3.4.1 Python Library API

Many of the NilmTools programs share functionality that is also available as Python modules. Two modules are detailed here. The first, `nilmtools.filter`, provides routines geared towards building command-line tools that accept arguments, and filter selected data from one source stream to one destination stream. Using it can help reduce the quantity of code needed to accomplish data processing tasks.

The second, `nilmtools.math`, provides some miscellaneous mathematical functions that are used by NilmTools filters, made available for external use.

3.4.1.1 Module `nilmtools.filter`

class `nilmtools.filter.`**`Filter`**`(`*parser_description = None*`)`

Main filter interface. This class assists with the parsing of command-line arguments followed by the running of a filter routine. Filters take data from a source stream, process it through a function, and write to a destination stream. When implemented with this module, the filter only processes data intervals that fall within the user-specified range, are present in the source stream, and are *not* present in the destination stream.

The source and destination streams can be located on different NilmDB servers.

[4]Not to be confused with the single `nilmtool` program, provided by the main NilmDB package.

Typical usage of this class to create a filter is as follows:

```
# Create filter interface
f = nilmtools.filter.Filter()

# Set up the basic command-line argument parsing and parse arguments
parser = f.setup_parser("My Filter")
args = f.parse_args()

# Run a custom filter function on chunks of the data
f.process_numpy(my_filter, args = args)
```

If *parser_description* is not *None*, it is automatically passed to `setup_parser`, and `parse_args` is run. This shortcut is generally not recommended, as it removes the ability to add additional arguments to the command-line interface.

Member functions and properties are described below. The properties are available after calling **`parse_args()`**.

property **`src`**

`nilmtools.filter.`**`StreamInfo`** object for the source stream.

property **`dest`**

`nilmtools.filter.`**`StreamInfo`** object for the destination stream.

property **`client_src`**

`nilmdb.client.numpyclient.`**`NumpyClient`** corresponding to the NilmDB server for the source stream.

property **`client_dest`**

`nilmdb.client.numpyclient.`**`NumpyClient`** corresponding to the NilmDB server for the destination stream.

property **`start`**

Starting timestamp for the filter

property **`end`**

Ending timestamp for the filter

Argument	Description
`-u` *or* `--url`	NilmDB server URL.
`-U` *or* `--dest-url`	NilmDB destination server URL.
`-D` *or* `--dry-run`	Just print intervals that would be processed, then exit after parsing arguments.
`--force-metadata`	Force metadata to be overwritten when **`check_dest_metadata()`** is used.
`-s` *or* `--start`	Starting timestamp for intervals
`-e` *or* `--end`	Starting timestamp for intervals
srcpath	Path of source stream
destpath	Path of destination stream

Table 3-28: Default command line parameters for the `Filter` library. See the documentation of `nilm-copy` (Section 3.4.2.2) for an example of a complete program with these arguments.

`setup_parser`(*description = "Filter Data"*, *skip_paths = False*) → `argparse.`**`ArgumentParser`**

Set up a `argparse.ArgumentParser` to parse the default set of command-line arguments. These are described in Table 3-28. Most of the arguments have default values; for more information, see the documentation of a tool such as `nilm-copy` (Section 3.4.2.2).

description is used as the description in the parser's help output. If *skip_paths* is True, the positional arguments `srcpath` and `destpath` are omitted. However, both arguments must be manually added to the parser for the remaining **`Filter`** functions to work correctly. This option is provided for filters that wish to reorder or add additional arguments.

This function returns the parser object, which can be further modified. For example, a caller can add an additional argument as follows:

```
# Set up the basic filter parser, and add additional arguments
parser = f.setup_parser("My Filter")
parser.add_argument("-z", "--size", action = "store", type = float,
                    default = 25, help = "Threshold size")
```

parse_args(*argv = None*) → `argparse.`**Namespace**

Parse the command-line arguments. If *argv* is *None*, the standard arguments in `sys.argv[1:]` are used. Returns the argument object returned by the `argparse.parse_args` function.

If the `--dry-run` argument was specified, this prints the intervals that would have been processed and exits.

set_args(*url*, *dest_url*, *srcpath*, *destpath*, *start*, *end*)

Instead of using **setup_parser()** and **parse_args()**, the parameters needed by the remaining functions in this class can be directly set using this method, for more complicated use cases.

check_dest_metadata(*data*)

When filtering streams, one may wish to add metadata to the destination stream to indicate the source and filter parameters. This can be useful to later determine how the destination stream was created, or to ensure that a single stream does not mix data from multiple sources or with incompatible parameters.

This function ensures that all of the key/value pairs in *data* are set in the metadata of the destination stream. If they do not exist or have empty values, they are created. If there is a conflict, that is, if a key in *data* has a different value than the same key in the metadata, an exception is raised.

The command-line option `--force-metadata`, created by **setup_args()**, can be used to skip the conflict check, in which case the destination stream metadata is unconditionally set to `data`.

intervals() → *generator*

Returns a generator that yields an **Interval** object (Section 3.2.2.5) for every time interval that this filter should process; that is, all intervals that are present in the source but not yet present in the destination.

process_numpy(*function*, *args = None*, *rows = 100000*, *intervals = None*)

Perform the main filtering operation. Filtering involves extracting chunks of data from the source stream as NumPy arrays and passing them to *function*, which processes the data and inserts the result into the destination stream.

function is a user-provided callback function with the same signature as `nilmtools.filter.`**`example_callback_function()`**. It accepts a NumPy array of data from the source, and can insert data into the destination stream. See the documentation of that function on page 177 for details on how it should be implemented.

args is additionally passed through to the function. A tuple or list should be provided if multiple additional arguments are required.

The parameter *rows* provides a soft limit on the number of rows of data passed to each invocation of *function*. Note that *function* returns a value indicating how many rows it processed. If not all rows were processed, subsequent calls will have new data appended to the remaining old data, and so the total number of rows passed per call may exceed the *rows* parameter.

If *intervals* is provided, it is used as the list of intervals for which to process data. If *None*, the method **`intervals()`** is used, which will return all intervals of data present in the source stream and not present in the destination stream.

class `nilmtools.filter.`**`StreamInfo`**(*url*, *info*)

Class representing information about a stream. *url* is the host URL of the NilmDB server, used only for display purposes. *info* is a list object, as returned by the client library method `stream_list`, documented on page 124. It is internally parsed into a series of member variables as follows:

property **path**

Path of the stream.

property **layout**

Layout string, e.g. `uint16_6`.

property **layout_type**

Layout type, e.g. `uint16`.

property **layout_count**

Layout count, e.g. `6`.

property **total_count**

Total columns of data, including timestamp; always `layout_count` + 1.

property **timestamp_min**

Earliest start time of any interval in the stream.

property **timestamp_max**

Latest end time of any interval in the stream.

property **rows**

Total rows of data in this stream.

property **seconds**

Total amount of time covered by the intervals of this stream, in seconds.

exception `nilmtools.filter.`**`ArgumentError`**`()`

Base exception: `Exception`

Exception raised due to certain errors in processing command-line arguments, such as identical source and destination streams.

exception `nilmtools.filter.`**`MissingDestination`**(*args*, *src*, *dest*)

Base exception: `Exception`

Exception raised when processing arguments, if the destination path specified to the filter is not found. *args* is the `argparse.`**`Namespace`** object containing the parsed command-line arguments. *src* and *dest* are `nilmtools.filter.`**`StreamInfo`** objects containing the known information about the source and destination streams. The intention is that the user program can use `src.layout_type` and `src.layout_count`, for example, to recommend how the missing destination stream should be created.

`nilmtools.filter.`**`example_callback_function`**(*data*, *interval*, *args*, *insert_func*, *final*) → *int*

This function is not intended to be used directly, but serves as documentation for how the *function* parameter to `nilmtools.filter.Filter().`**`process_numpy()`** should be defined and behave.

When called, the *data* argument is a 2-dimensional NumPy array containing data from the source stream, in the homogeneous format used by the client library method `stream_extract_numpy`, documented on page 128. The number of rows in the array is variable; it will be at least one, with no fixed upper limit, but is typically around the *rows* value passed to **`process_numpy()`**.

interval is an **`Interval`** object, representing the complete interval to which this data belongs. There may also be other data in the same interval, both before and after the given *data* array. All of the timestamps in *data* are guaranteed to fall within the range $[interval.start \rightarrow interval.end\rangle$.

args is exactly the *args* argument passed to **`process_numpy()`**, and can be used to provide additional data to the callback function.

insert_func is a function that should be called to insert data into the destination stream. It takes a single argument, a 2-dimensional NumPy array in the same format as the original *data* argument. All of the timestamps in this array must

fall within the current *interval*, and should also fall within the minimum and maximum timestamps seen in the original *data*, to avoid overlap with subsequent chunks.

final indicates whether or not this is the final chunk of data that will be passed for the current *interval*. If *True*, no more calls will be made for this interval. If *False*, this function will be called again, with any rows that were not processed (based on its return value, as described below).

The return value of this function is an integer representing the number of rows that were processed from the *data* array. Unprocessed rows will be saved and provided again in a subsequent call, with new data appended. This is intended to assist filter operations that need to work on blocks of data; partial blocks can be left unprocessed so that they are provided later when more data follows.

Generally, the intervals being inserted into the destination stream will match the *interval* arguments to this function. As a special case, if unprocessed data remains in the source stream after *final* is *True*, the interval being inserted will end at the timestamp of the first unprocessed data point. This is to ensure that the unprocessed data can be reconsidered when the filter is run again.

An example callback function that copies data directly from the input stream to the output stream, but works on blocks of 100 rows at a time, could be implemented as follows:

```
def my_filter(data, interval, args, insert_func, final):
    (rows, cols) = data.shape
    start_row = 0
    end_row = start_row + 100
    while end_row <= rows:
        print "processing row", start_row, "through", end_row
        block = data[start_row:end_row, :]
        insert_func(block)
        start_row += 100
        end_row += 100
    print "processed", start_row, "total rows"
    return start_row
```

`nilmtools.filter.`**`get_stream_info`**(*client*, *path*) → `nilmtools.filter.`**`StreamInfo`**

Return a **`StreamInfo`** object for the stream at the given *path*. *client* is a `nilmdb.client.client.Client` object associated with the NilmDB server.

3.4.1.2 Module `nilmtools.math`

`nilmtools.math.`**`sfit4`**(*data*, *fs*) → (*A*, *f0*, *phi*, *C*)

Compute the least-squares best fit of a 4-parameter sine wave to the given data vector, using the technique recommended by IEEE Std 1241-2010 Annex B.2 [53] and described in Section 4.3.1. Specifically, given the input data vector *data* and the sampling rate f_s in Hz, compute and return a tuple with the optimal parameters $(A,\ f_0,\ \phi,\ C)$ to the equation

$$x[n] = A \cdot \sin\left(2\pi n \frac{f_0}{f_s} + \phi_0\right) + C$$

such that the value $\sum \left(x[n] - data[n]\right)^2$ is minimized.

`nilmtools.math.`**`peak_detect`**(*data*, *delta = 0.1*) → *list*

Simple minima and maxima detection algorithm. Given the input data vector *data* and a threshold δ, the algorithm alternates between finding local minima and local maxima as it sweeps through *data*.

When searching for maxima, the algorithm tracks the largest value seen so far, $data[n] = P$. When a subsequent value drops below $P - \delta$, the tuple *(n, P, True)* is appended to an output list, indicating a maximum. Minima are located analogously, and the tuple *(n, P, False)* is appended to indicate each minimum. The combined output list is returned.

For example:

```
peaks = nilmtools.math.peak_detect(data, 0.1)
for (n, p, is_max) in peaks:
    if is_max:
        print "found local maximum", p, "at index", n
    else:
        print "found local minimum", p, "at index", n
```

3.4.2 Command Line Reference

NilmTools provides a variety of programs and filters with command-line interfaces, all beginning with the prefix `nilm-`. Each of these programs is described in the following sections.

Many of the NilmTools programs are filters that process input from one or more source streams into a destination stream. Only regions of time that are present in the source, and not yet present in the destination, are processed. These programs can therefore be re-run with the same command-line arguments multiple times, and they will only process the newly available data each time.

The conventions used for the command-line documentation are the same as those described in Section 3.2.3. Many of the programs support arguments that represent a NilmDB timestamp. This timestamp is specified as a free-form string as supported by the `parse_time` client library function described in Section 3.2.2.4, with examples shown in Table 3-19 on page 133.

3.4.2.1 `nilm-cleanup` – Clean up old data from streams

Usage

```
nilm-cleanup [-h] [-v] [-u URL] [-y] [-e] CONFIGFILE
```

Description

Clean up old data from streams, using a configuration file to specify which data to remove. The configuration file is a text file in the following format:

```
[/stream/path]
keep = 3w         # keep up to 3 weeks of data
rate = 8000       # optional, used for the --estimate option
decimated = false # whether to delete decimated data too

[*/prep]
keep = 3.5m       # or 2520h or 105d or 15w or 0.29y
```

Stream paths are specified inside square brackets (`[]`) and are followed by configuration keywords for the matching streams. Paths can contain wildcards.

Supported keywords are:

`keep`

How much data to keep. Supported suffixes are `h` for hours, `d` for days, `w` for weeks, `m` for months, and `y` for years.

`rate` *(default:* automatic)

Expected data rate. Only used by the `--estimate` option. If not specified, the rate is guessed based on the existing data in the stream.

`decimated` *(default:* `true`)

If `true`, delete decimated data too. For stream path `/A/B`, this includes any stream matching the wildcard `/A/B~decim*`. If specified as `false`, no special treatment is applied to such streams.

Arguments

`-u` *`URL`*`, --url` *`URL`* *(default:* `http://localhost/nilmdb/`)

NilmDB server URL.

`-y, --yes`

Actually remove the data. By default, `nilm-cleanup` only prints what it *would* have removed, but leaves the data intact.

`-e, --estimate`

Instead of removing data, print an estimated report of the maximum amount of disk space that will be used by the cleaned-up streams. This uses the on-disk size of the stream layout, the estimated data rate, and the space required by decimation levels. Streams not matched in the configuration file are not included in the total.

`CONFIGFILE`

Path to the configuration file.

Notes

The value `keep` is a maximum amount of data, not a cutoff time. When cleaning data, the oldest data in the stream will be removed, until the total remaining amount of data is less than or equal to `keep`. This means that data older than `keep` will remain if insufficient newer data is present; for example, if new data ceases to be inserted, old data will cease to be deleted.

3.4.2.2 `nilm-copy` – Copy data between streams

Usage

```
nilm-copy [-h] [-v] [-u URL] [-U DESTURL] [-D] [-F] [-n]
          [-s TIME] [-e TIME] SRCPATH DESTPATH
```

Description

Copy data and metadata from one stream to another. The source and destination streams can reside on different servers. Both streams must have the same layout.

Arguments

`-u URL, --url URL` *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL for the source stream.

`-U DESTURL, --dest-url DESTURL` *(default:* same as `URL`*)*

NilmDB server URL for the destination stream. If unspecified, the same URL is used for both source and destination.

`-D, --dry-run`

Just print intervals that would be processed, and exit.

`-F, --force-metadata`

Metadata is copied from the source to the destination. By default, an error

is returned if the destination stream metadata conflicts with the source stream metadata. Specify this flag to always overwrite the destination values with those from the source stream.

`-n, --nometa`

Don't copy or check metadata at all.

`-s` *TIME*`, --start` *TIME* *(default:* `min`)

Starting timestamp of data to copy (free-form, inclusive).

`-e` *TIME*`, --end` *TIME* *(default:* `max`)

Ending timestamp of data to copy (free-form, noninclusive).

SRCPATH

Path of the source stream (on the source server).

DESTPATH

Path of the destination stream (on the destination server).

3.4.2.3 `nilm-copy-wildcard` – Copy multiple streams

Usage

```
nilm-copy-wildcard [-h] [-v] [-u URL] [-U DESTURL] [-D] [-F] [-n]
                   [-s TIME] [-e TIME] PATHS [...]
```

Description

Copy data and metadata, from multiple streams, between two servers. Similar to `nilm-copy`, except:

- Wildcards and multiple paths are supported in the stream names.
- Streams must always be copied between two servers.
- Stream paths must match on the source and destination server.

- If a stream does not exist on the destination server, it is created with the correct layout automatically.

This is intended for moving data from one NilmDB server to another.

Arguments

Most arguments are identical to those of `nilm-copy`; see Section 3.4.2.2 for details.

PATHS

Path(s) to copy from the source server to the destination server. Wildcards are accepted.

Example

```
$ nilm-copy-wildcard -u http://bucket/nilmdb \            ↩ (continued)
                     -U http://pilot/nilmdb /bp/startup*
Source URL: http://bucket/nilmdb/
  Dest URL: http://pilot/nilmdb/
Creating destination stream /bp/startup/info
Creating destination stream /bp/startup/prep-a
Creating destination stream /bp/startup/prep-a~decim-4
Creating destination stream /bp/startup/prep-a~decim-16
... etc
```

3.4.2.4 `nilm-decimate` – Decimate a stream one level

Usage

```
nilm-decimate [-h] [-v] [-u URL] [-U DESTURL] [-D] [-F]
              [-s TIME] [-e TIME] [-f FACTOR] SRCPATH DESTPATH
```

Description

Decimate the stream at *SRCPATH* and write the output to *DESTPATH*. The decimation operation is described in Section 2.4.1; in short, every *FACTOR* rows in the source are consolidated into one row in the destination, by calculating the mean, minimum, and maximum values for each column.

This program detects if the stream at *SRCPATH* is already decimated, by the presence of a `decimate_source` metadata key. If present, subsequent decimations take the existing mean, minimum, and maximum values into account, and the output has the same number of columns as the input. Otherwise, for the first level of decimation, the output has three times as many columns as the input.

See also `nilm-decimate-auto` (Section 3.4.2.5) for a simpler method of decimating a stream by multiple levels.

Arguments

`-u` *URL*`, --url` *URL* *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL for the source stream.

`-U` *DESTURL*`, --dest-url` *DESTURL* *(default:* same as `URL`*)*

NilmDB server URL for the destination stream. If unspecified, the same URL is used for both source and destination.

`-D, --dry-run`

Just print intervals that would be processed, and exit.

`-F, --force-metadata`

Overwrite destination metadata even if it conflicts with the values in the "metadata" section below.

`-s` *TIME*`, --start` *TIME* *(default:* `min`*)*

Starting timestamp of data to decimate (free-form, inclusive).

`-e` *TIME*`, --end` *TIME* *(default:* `max`*)*

Ending timestamp of data to decimate (free-form, noninclusive).

`-f` *FACTOR*`, --factor` *FACTOR* *(default:* `4`*)*

Set the decimation factor. For a source stream with n rows, the output stream will have $n/$`FACTOR` rows.

SRCPATH

Path of the source stream (on the source server).

DESTPATH

Path of the destination stream (on the destination server).

Metadata

The destination stream has the following metadata keys added:

`decimate_source`

The source stream from which this data was decimated.

`decimate_factor`

The decimation factor used.

3.4.2.5 `nilm-decimate-auto` – Decimate a stream completely

Usage

`nilm-decimate-auto` `[-h] [-v] [-u` *`URL`*`] [-F] [-f` *`FACTOR`*`]` *`PATH`* `[...]`

Description

Automatically create multiple decimation levels using from a single source stream, continuing until the last decimated level contains fewer than 500 rows total. Decimations are performed using `nilm-decimate` (Section 3.4.2.4).

Wildcards and multiple paths are accepted. Destination streams are automatically named based on the source stream name and the total decimation factor; for example, `/test/raw~decim-4`, `/test/raw~decim-16`, etc. Streams containing the string "`~decim-`" are ignored when matching wildcards.

This is the recommended program to use when decimating data for use by the NILM Manager.

Arguments

`-u URL, --url URL` *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL for the source and destination streams.

`-F, --force-metadata`

Overwrite destination metadata even if it conflicts with the values in the "metadata" section above.

`-f FACTOR, --factor FACTOR` *(default:* `4`*)*

Set the decimation factor. Each decimation level will have 1/`FACTOR` as many rows as the previous level.

PATH `[...]`

One or more paths to decimate. Wildcards are accepted.

3.4.2.6 `nilm-insert` – Insert data from an external source

Usage

```
nilm-insert [-h] [-v] [-u URL] [-D] [-s] [-m SEC] [-r RATE | -d]
            [-l | -f] [-o SEC] [-O SEC] PATH [INFILE ...]
```

Description

Insert a large amount of text-formatted data from an external source like eth-stream. This is a higher-level tool than `nilmtool insert` in that it attempts to intelligently manage timestamps. The general concept is that it tracks two timestamps:

1. The *data* timestamp is the precise timestamp corresponding to a particular row of data, and is the timestamp that gets inserted into the database. It increases by `data_delta` for every row of input.

   `data_delta` can come from one of two sources. If `--delta` is specified, it is

pulled from the first column of data. If `--rate` is specified, `data_delta` is set to a fixed value of 1/`RATE`.

2. The *`clock`* timestamp is the less precise timestamp that gives the absolute time. It can come from two sources. If `--live` is specified, it is pulled directly from the system clock. If `--file` is specified, it is extracted from the input file every time a new file is opened for read, and from comments that appear in the files.

Small discrepancies between *`data`* and *`clock`* are ignored. If the *`data`* timestamp ever differs from the *`clock`* timestamp by more than `max_gap` seconds:

- If *`data`* is running behind, there is a gap in the data, so the timestamp is stepped forward to match *`clock`*.
- If *`data`* is running ahead, there is overlap in the data, and an error is returned. If `--skip` is specified, then instead of returning an error, data is dropped and the remainder of the current file is skipped.

Arguments

`-u` *`URL`*`, --url` *`URL`* *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL.

`-D, --dry-run`

Parse files and print information, but don't insert any data. Useful for verification before making changes to the database.

`-s, --skip`

Skip the remainder of input files if the *`data`* timestamp runs too far ahead of the *`clock`* timestamp. Useful when inserting a large directory of existing files with inaccurate timestamps.

`-m SEC, --max-gap SEC` *(default:* 10.0)

Maximum discrepancy between the *`clock`* and *`data`* timestamps.

Data timestamp

`-r RATE, --rate RATE` *(default:* 8000.0)

`data_delta` is constant 1/`RATE` (in Hz).

`-d, --delta`

`data_delta` is provided as the first number on each input line.

Clock timestamp

`-l, --live`

Use the live system time for the *`clock`* timestamp. This is most useful when piping in data live from a capture device.

`-f, --file`

Use filename and file comments for the *`clock`* timestamp. This is most useful when reading previously saved data.

`-o SEC, --offset-filename SEC` *(default:* −3600.0)

Offset to add to timestamps in filenames, when using `--file`. The default accounts for the existing practice of naming capture files based on the end of the hour in which they were recorded. The filename timestamp plus this offset should equal the time that the first row of data in the file was captured.

`-O SEC, --offset-comment SEC` *(default:* 0.0)

Offset to add to timestamps in comments, when using `--file`. The comment timestamp plus this offset should equal the time that the next row of data was captured.

Path and Input

PATH

Path of the stream into which to insert data. The layout of the path must match the input data.

INFILE `[...]` *(default:* standard input)

Input data filename(s). Filenames ending with `.gz` are transparently decompressed as they are read. The default is to read the input from `stdin`.

3.4.2.7 `nilm-median` – Sliding window median filter

Usage

```
nilm-median [-h] [-v] [-u URL] [-U DESTURL] [-D] [-F]
            [-s TIME] [-e TIME] [-z ROWS] [-d] SRCPATH DESTPATH
```

Description

Compute the median value over sliding windows of the source stream. Store either the median, or the difference between the original data and the median, in the destination stream.

Arguments

`-u` *URL*`, --url` *URL* *(default:* `http://localhost/nilmdb/`)

NilmDB server URL for the source stream.

`-U` *DESTURL*`, --dest-url` *DESTURL* *(default:* same as `URL`)

NilmDB server URL for the destination stream. If unspecified, the same URL is used for both source and destination.

`-D, --dry-run`

Just print intervals that would be processed, and exit.

`-F, --force-metadata`

Overwrite destination metadata even if it conflicts with the values in the "metadata" section below.

`-s` *TIME*`, --start` *TIME* *(default:* `min`)

Starting timestamp of data to filter (free-form, inclusive).

`-e` *TIME*`, --end` *TIME* *(default:* `max`)

Ending timestamp of data to filter (free-form, noninclusive).

`-z` *ROWS*`, --size` *ROWS* *(default:* 25)

Size of the sliding window and the median filter, in rows.

`-d, --difference`

Store difference rather than filtered values.

SRCPATH

Path of the source stream (on the source server).

DESTPATH

Path of the destination stream (on the destination server).

Metadata

The destination stream has the following metadata keys added:

`median_filter_source`

The source stream from which this data was created.

`median_filter_size`

The size of the window used in the filter.

`median_filter_difference`

`True` if the `--difference` option was used, `False` otherwise.

3.4.2.8 `nilm-pipewatch` – Pipe data between processes

Usage

`nilm-pipewatch` `[-h] [-v] [-d] [-l` *`FILE`*`] [-t` *`SECONDS`*`]` *`GENERATOR CONSUMER`*

Description

Pipe data between a "generator" and "consumer" process. This is similar to running "`generator | consumer`" from a command line, but is more robust in case of error conditions. This is intended to be executed frequently from a job scheduler such as `cron`, and will exit if another copy is already running. If the generator or consumer returns an error, or if the generator stops sending data for a while, it will exit.

Typical usage is with `ethstream` as a generator and `nilm-insert` as the consumer.

Arguments

`-l` *`FILENAME`*`, --lock` *`FILENAME`* *(default:* `/tmp/nilm-pipewatch.lock`*)*

Lock file for detecting running instances. Only one instance of `nilm-pipewatch` using the same lock file will run at any given time; other instances will exit at startup.

`-d, --daemon`

Run in the background. If specified, the pipeline is executed in the background as soon as the lock is acquired, and no further output is printed.

`-t` *`SECONDS`*`, --timeout` *`SECONDS`* *(default:* 30.0*)*

Restart the pipeline if the generator fails to send any output for this long. Useful for detecting a "stuck" acquisition process.

GENERATOR

Command and arguments to generate data, provided as a single string that will be executed through a shell.

CONSUMER

Command and arguments to consume data, provided as a single string that will be executed through a shell.

Example

A suitable `crontab` entry might be as follows:

```
*/5 * * * * /home/nilmdb/data/capture.sh >/dev/null
```

where `capture.sh` contains

```
#!/bin/bash
exec nilm-pipewatch --daemon --lock "/tmp/capture.lock" --timeout 30 \
   "ethstream --nerdjack --address 192.168.1.209 -n 9 --rate 8000"   \
   "nilm-insert --max-gap 10 --rate 8000 --live /sharon/raw"
```

3.4.2.9 `nilm-prep` – Spectral envelope preprocessor

Usage

```
nilm-prep [-h] [-v] [-u URL] [-U DESTURL] [-D] [-F] [-s TIME] [-e TIME]
          [-c COLUMN] [-n NHARM] [-N NSHIFT] [-r DEG | -R RAD ]
          SRCPATH SINEPATH DESTPATH
```

Description

Perform the spectral envelope harmonic coefficient calculation described in Section 4.3.3. Two source streams are provided, one with the raw current data and one with marked zero crossings, typically created by `nilm-sinefit` (Section 3.4.2.10). The filter processes regions of time that are present in both source streams, and not present in the destination stream.

General Arguments

`-u URL, --url URL` *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL for the source stream.

`-U DESTURL, --dest-url DESTURL` *(default:* same as `URL`*)*

NilmDB server URL for the destination stream. If unspecified, the same URL is used for both source and destination.

`-D, --dry-run`

Just print intervals that would be processed, and exit.

`-F, --force-metadata`

Overwrite destination metadata even if it conflicts with the values in the "metadata" section below.

`-s TIME, --start TIME` *(default:* `min`*)*

Starting timestamp of data to filter (free-form, inclusive).

`-e TIME, --end TIME` *(default:* `max`*)*

Ending timestamp of data to filter (free-form, noninclusive).

Preprocessor Arguments

`-c COLUMN, --column COLUMN`

Column number in `SRCPATH` to use for the raw data. The first data column is 1.

`-n NHARM, --nharm NHARM` *(default:* 4*)*

Number of odd harmonics N_{harm} to compute and store. For example, $N_{\text{harm}} = 2$ will store P_1, Q_1, P_3, and Q_3.

`-N NSHIFT, --nshift NSHIFT` *(default:* 1)

Number of shifted FFTs N_{shift} to compute, per period of the raw data. If the input frequency is 60 Hz, the data rate of the preprocessor output is $N_{\text{shift}} \cdot 60$ Hz.

Note that the calculation used by the similar Kalman-filter preprocessor, described in [54], is equivalent to $N_{\text{shift}} = 2$.

`-r DEG, --rotate DEG` *(default:* 0.0°)

Apply the additional rotation ϕ_{extra} to the FFT output, in degrees. Typically used to account for known phase offset between voltage and current. This is equivalent to adding a lag of ϕ_{extra} degrees to the zero crossing data.

This is also useful for three-phase systems. For example, the zero crossings can be calculated once with `nilm-sinefit` on φA voltage. Then, `nilm-prep` can be run on φA, φB, and φC currents using rotations of 0, 120, and 240 degrees. The order in which to apply these shifts will depend on the phase ordering in the measured system.

`-R RAD, --rotate-rad RAD` *(default:* 0 rad)

Like `--rotate`, except specified in radians instead of degrees.

SRCPATH

Path of the raw source stream, for example, `/foo/raw`.

SINEPATH

Path of the sinefit source stream, for example, `/foo/sinefit`.

DESTPATH

Path of the prep output, for example, `/foo/prep`. The destination stream must have $2 \cdot N_{\text{harm}}$ columns.

Metadata

The destination stream has the following metadata keys added:

`prep_raw_source`

The source stream of the raw data from which these envelopes were calculated.

`prep_sinefit_source`

The source stream of the marked zero crossings used for this data.

`prep_column`

The column number of the raw data in the raw data source.

`prep_rotation`

The applied rotation ϕ_{extra} for this data, in radians.

`prep_nshift`

The number of shifted FFTs N_{shift} for this data.

3.4.2.10 `nilm-sinefit` – Sinusoid fitting

Usage

```
nilm-sinefit [-h] [-v] [-u URL] [-U DESTURL] [-D] [-F] [-s TIME] [-e TIME]
             [-c COLUMN] [-f FREQ] [-m MIN_FREQ] [-M MAX_FREQ]
             [-a MIN_AMP] SRCPATH DESTPATH
```

Description

Perform the 4-parameter sinefit fit calculation described in Section 4.3.2. Given a rough estimate of the frequency, this filter looks at successive windows of approximately 3 - 4 periods of the input waveform. For each window, it computes the least-squares best fit sinusoid of the form:

$$x[n] = A \cdot \sin\left(2\pi n \frac{f_0}{f_s} + \phi_0\right) + C \tag{3.1}$$

At each of the positive zero crossings ($\phi = 0$) of the fit, the timestamped values f_0, A, and C corresponding to the subsequent period are stored.

The output stream will have one row of output per period of the input stream. The window sliding algorithm is designed to ensure that zero crossings do not occur near the window boundaries in order to reduce error.

The fitted sinusoid is checked against frequency and amplitude limits. If the fit falls outside the given bounds, no data points are inserted into the destination stream for that particular window.

General Arguments

`-u` *`URL`*`, --url` *`URL`* *(default:* `http://localhost/nilmdb/`*)*

NilmDB server URL for the source stream.

`-U` *`DESTURL`*`, --dest-url` *`DESTURL`* *(default:* same as `URL`*)*

NilmDB server URL for the destination stream. If unspecified, the same URL is used for both source and destination.

`-D, --dry-run`

Just print intervals that would be processed, and exit.

`-F, --force-metadata`

Overwrite destination metadata even if it conflicts with the values in the "metadata" section below.

`-s` *`TIME`*`, --start` *`TIME`* *(default:* `min`*)*

Starting timestamp of data to filter (free-form, inclusive).

`-e` *`TIME`*`, --end` *`TIME`* *(default:* `max`*)*

Ending timestamp of data to filter (free-form, noninclusive).

Sinefit Arguments

`-c` *`COLUMN`*`, --column` *`COLUMN`*

Column number in `SRCPATH` to use for the source data. The first data column is 1.

`-f` *`FREQ`*`, --frequency` *`FREQ`* *(default:* 60.0)

Rough estimate of the input frequency f_{est}, used only to determine the size of the window to analyze and to set defaults for the minimum and maximum frequency. Given an average sampling rate f_s of the input data, the sine wave fit is performed against windows of $N = 3.5 \cdot f_s / \texttt{FREQ}$ points.

`-m` *`MIN_FREQ`*`, --min-freq` *`MIN_FREQ`* *(default:* $f_{\text{est}}/2$)

Minimum valid frequency f_0 of the fitted sinusoid.

`-m` *`MAX_FREQ`*`, --max-freq` *`MAX_FREQ`* *(default:* $f_{\text{est}} \cdot 2$)

Maximum valid frequency f_0 of the fitted sinusoid.

`-a` *`MIN_AMP`*`, --min-amp` *`MIN_AMP`* *(default:* 20.0)

Minimum valid amplitude A of the fitted sinusoid.

`SRCPATH`

Path of the raw source stream, for example, `/foo/raw`.

`DESTPATH`

Path of the fitted output parameters, for example, `/foo/sinefit`.

Metadata

The destination stream has the following metadata keys added:

`sinefit_source`

The source stream of the raw data used to fit these parameters.

`sinefit_column`

The column number used from the source stream.

3.4.2.11 `nilm-trainola` – Perform exemplar matching

Usage

`nilm-trainola` `[-h] [-v] <json-config-dictionary>`

Description

Perform exemplar-based transient event matching on a stream. Exemplars specify short signal segments, and the input stream is searched for regions that match the shape and amplitude of the exemplars. Matching locations are identified in an output stream.

Multiple columns in the input stream and exemplars can be simultaneously checked for matches, with weightings based on relative magnitude.

Arguments

Due to the complex nature of input and exemplar specifications, this tool requires a single argument containing a JSON-formatted dictionary of parameters. It is not intended for direct manual execution from the command line, but is typically run through the NILM Manager (Section 2.4).

The top-level parameter dictionary contains keys and values, some of which contain additional dictionaries. Each key is documented below, and an example follows.

Top-level Dictionary

`url` *(string)*

NilmDB server URL for the main input and output stream.

`stream` *(string)*

Stream path of the input stream, e.g. `/sharon/prep-a`.

`dest_stream` *(string)*

Output stream. If matching N exemplars, the output stream must contain N columns. When one or more exemplar is matched, a timestamped row is inserted into the output stream. Exemplars that were matched at this timestamp are indicated with a 1 in their respective column, while exemplars that did not match are indicated with 0. The mapping between exemplars and columns is defined in the `exemplars` list.

`start` *(integer)*

Starting timestamp of the interval to search in the input stream, in NilmDB units (microseconds since epoch).

`end` *(integer)*

Ending timestamp of the interval to search in the input stream, in NilmDB units (microseconds since epoch).

`columns` *(list)*

A list of dictionaries describing the columns of the input stream. The format of each dictionary is described in the "column dictionary" section below. Each named column in the exemplars must have a corresponding named column in the input stream. Not all input stream columns need to be identified, if they are not referenced by exemplars.

`exemplars` *(list)*

A list of dictionaries describing the exemplars. The format of each dictionary is described in the "exemplar dictionary" section below. At least one exemplar must be provided.

Column Dictionary

`name` *(string)*

An arbitrary string describing the data in this column, e.g. `P1`.

`index` *(integer)*

The corresponding column number. The first non-timestamp column is denoted 0.

Exemplar Dictionary

`name` *(string)*

The name of the exemplar, e.g. "Pump 1 On".

`dest_column` *(integer)*

The column number of the output stream, into which matches of this exemplar will be marked. The first non-timestamp column is denoted 0. Multiple exemplars can use the same output column number; for example, this could be used for cases when two different exemplars should be identified as the same event.

`url` *(string)*

NilmDB server URL for the stream containing the exemplar.

`stream` *(string)*

Stream path containing the exemplar.

`start` *(integer)*

Starting timestamp of the interval to search in the input stream, in NilmDB units (microseconds since epoch).

`end` *(integer)*

Ending timestamp of the interval to search in the input stream, in NilmDB units (microseconds since epoch).

`columns` *(list)*

A list of dictionaries describing the columns of the exemplar stream. The format of each dictionary is described in the "column dictionary" section above. Columns included in this list are used when matching the exemplar; other columns are ignored.

Example Parameters

```
{   "url": "http://bucket.mit.edu/nilmdb",
    "dest_stream": "/sharon/prep-a-matches",
    "stream": "/sharon/prep-a",
    "start": 1366111383280463,
    "end": 1366126163457797,
    "columns": [ { "name": "P1", "index": 0 },
                 { "name": "Q1", "index": 1 },
                 { "name": "P3", "index": 2 } ],
    "exemplars": [
        { "name": "Boiler Pump ON",
          "url": "http://bucket.mit.edu/nilmdb",
          "stream": "/sharon/prep-a",
          "start": 1366260494269078,
          "end": 1366260608185031,
          "dest_column": 0,
          "columns": [ { "name": "P1", "index": 0 },
                       { "name": "Q1", "index": 1 }
                     ]
        },
        { "name": "Boiler Pump OFF",
          "url": "http://bucket.mit.edu/nilmdb",
          "stream": "/sharon/prep-a",
          "start": 1366260864215764,
          "end": 1366260870882998,
          "dest_column": 1,
          "columns": [ { "name": "P1", "index": 0 },
                       { "name": "P3", "index": 2 }
                     ]
        }
    ]
}
```

Example Execution

The `nilm-trainola` process prints progress while it is identifying transients. Here, the parameters are stored in a file `params.js`, and passed by using the shell to read the contents of that file:

```
$ nilmtool create /sharon/prep-a-matches int8_4
$ nilm-trainola $(cat params.js)
Trainola 1.4.10
Source:
  /sharon/prep-a [P1,Q1,P3]
Destination:
  /sharon/prep-a-matches (4 columns)
Loading exemplar 0:
  "Boiler Pump ON" /sharon/prep-a [P1,Q1] 6827 rows, output column 0
Loading exemplar 1:
  "Boiler Pump OFF" /sharon/prep-a [P1,P3] 399 rows, output column 1
Processing intervals:
[ Tue, 16 Apr 2013 07:56:23.280463 -> Tue, 16 Apr 2013 09:16:03.457797 ]
  [07:56:23] matched 4 exemplars in 93174 rows
  [08:22:18] matched 0 exemplars in 100000 rows
  [08:50:06] matched 2 exemplars in 86443 rows
Done. Processed 2669.40 seconds per second.
```

Example Output

The contents of the output stream indicates the matches and their timestamps:

```
$ nilmtool extract -s min -e max /sharon/prep-a-matches
1366113634334940 0 1 0 0
1366113851543878 1 0 0 0
1366114213316598 0 1 0 0
1366114431740785 1 0 0 0
1366117451467920 0 1 0 0
1366117645674450 1 0 0 0
```

Chapter 4

The Sinefit Spectral Envelope Preprocessor

4.1 Introduction

For many electrical systems driven by ac sources, the "spectral envelope" representation of observed current and voltage signals has proven to be a widely useful and powerful metric for classification, diagnostics, and power quality measurement [1, 3, 12, 13, 54–56]. Spectral envelopes describe the harmonic content of the measured signals at integer multiples of the ac line frequency driving the monitored loads. Such loads exhibit behaviors that are synchronous with the line frequency. By extracting spectral envelopes, the preprocessor facilitates physically based analysis of power and current consumption.

The spectral envelope preprocessor has two primary tasks: phase and frequency estimation, and harmonic coefficient calculation. Existing versions of the spectral envelope preprocessor vary in their implementations. For phase and frequency estimation, common techniques include phase-locked loops [12,57], weighted least-squares estimators [58], and Kalman filters [54, 59]. To reduce computation load, existing implementations have utilized techniques such as analog multipliers and precomputed basis vectors [12, 54].

This chapter presents the Sinefit spectral envelope preprocessor, based on non-linear least-squares sinusoid fitting combined with the DFT. These techniques focus on accuracy and implementation flexibility, reflecting the growing availability of high performance computing resources. The preprocessor is implemented within the NilmDB framework presented in Chapter 2, allowing reuse and replacement of computation components for related or optimized calculations. Sinefit solves problems with existing preprocessors by providing more robust phase and frequency detection, accurate timestamping of spectral envelopes, and improved and quantifiable accuracy.

4.2 Spectral Envelopes

Spectral envelopes $a_k(t)$ and $b_k(t)$ are short-term averages of harmonic content, calculated over sliding windows of an input signal. Figure 4-1 demonstrates spectral envelopes as computed for an ac load. The first plot is the raw sampled input from a data acquisition board. The second shows the in-phase and quadrature components of the first harmonic (60 Hz) envelopes.

4.2.1 Definition

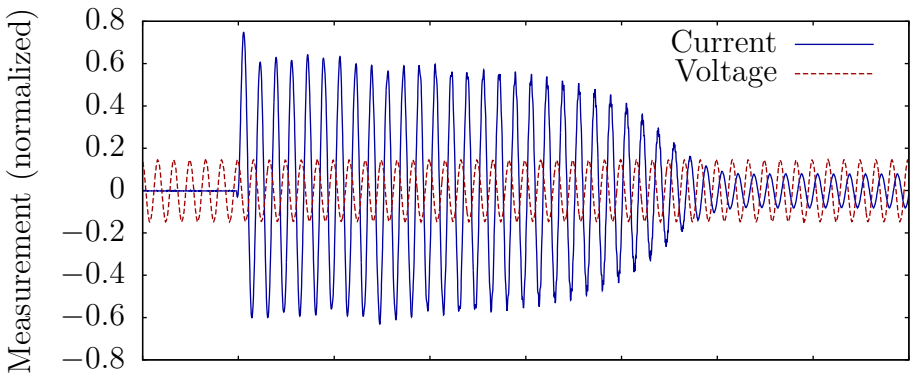
A periodic signal $f(t)$ with period $T = 1/f_0$ can be expressed in terms of its spectral content using the Fourier series:

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \sin(k2\pi f_0 t) + b_k \cos(k2\pi f_0 t)\right) \tag{4.1}$$

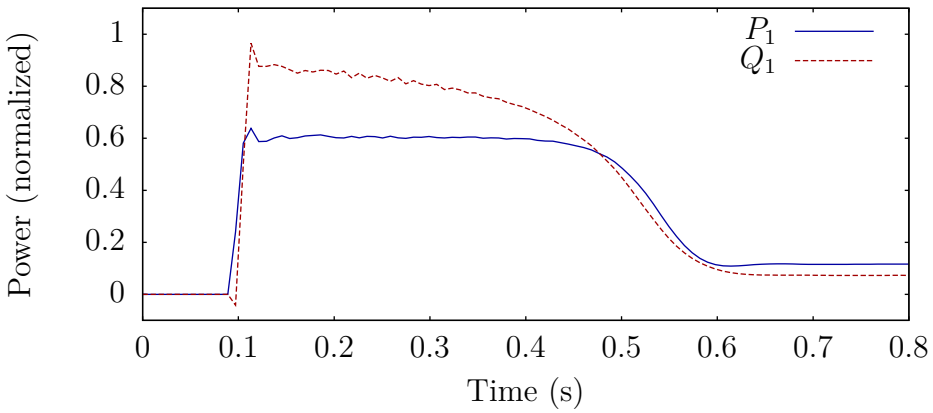
where the harmonic coefficients a_k and b_k are defined as:

$$a_k = \frac{2}{T} \int_0^T f(t) \cdot \sin(k(2\pi f_0 t))dt \tag{4.2}$$

$$b_k = \frac{2}{T} \int_0^T f(t) \cdot \cos(k(2\pi f_0 t))dt \tag{4.3}$$



(a) Raw ac voltage and current measurements



(b) Computed spectral envelopes

Figure 4-1: Example of computed spectral envelopes for an ac load.

For the NILM, we assume that the voltage and current signals $v(t)$ and $i(t)$ are *locally* periodic over one ac line cycle. For a discrete input $i[n]$ sampled at rate f_s, one period is of length $N = f_s/f_0$ samples, and we can compute harmonic coefficients as:

$$a_k = \frac{2}{N} \sum_{n=0}^{N-1} i[n] \cdot \sin(k(2\pi n/N)) \tag{4.4}$$

$$b_k = \frac{2}{N} \sum_{n=0}^{N-1} i[n] \cdot \cos(k(2\pi n/N)) \tag{4.5}$$

Here, k denotes the multiple of the line frequency to which a particular coefficient corresponds; for example, $k = 1$ corresponds to the 60 Hz component and $k = 3$ to the 180 Hz component.

Harmonic coefficients can also be calculated in complex form using the DFT, which is defined [60] as:

$$X_k = \mathcal{F}(x[n]) = \sum_{n=0}^{N-1} x[n] \cdot e^{-jk2\pi\eta/N} \tag{4.6}$$

Using Euler's formula, we can extract a_k and b_k in terms of the DFT as:

$$e^{-jw} = \cos(\omega t) - j \sin \omega(t) \tag{4.7}$$

$$a_k = -\frac{2}{N} \operatorname{imag}(X_k) \tag{4.8}$$

$$b_k = \frac{2}{N} \operatorname{real}(X_k) \tag{4.9}$$

where, as before, $N = f_s/f_0$.

The values of these coefficients are calculated for successive or sliding windows of the input signal, and the resulting time-varying harmonics $a_k[m]$ and $b_k[m]$ are the spectral envelopes.

Spectral envelopes can be extracted from any periodic or quasiperiodic input signal, and are typically computed for the current, using the voltage as the phase and frequency reference. When computed separately for voltage $v[n]$ and current $i[n]$, we denote the coefficients as a_{vk}, b_{vk}, a_{ik}, and b_{ik}. Then, first harmonic real and reactive power are:

$$P_1 = a_{v1} \cdot a_{i1} \qquad\qquad Q_1 = b_{v1} \cdot b_{i1} \tag{4.10}$$

For relatively "stiff" and harmonic-free utility voltage, a_{v1} and b_{v1} can be assumed to be constant, in which case:

$$P_1 \propto a_{i1} \qquad\qquad Q_1 \propto -b_{i1} \tag{4.11}$$

4.2.2 Phase Rotation

The coefficients a_k and b_k in (4.8) and (4.9) are calculated with sliding or successive windows over the input data. These windows must be phase-aligned with a reference, typically the utility voltage, such that $n = 0$ corresponds to, for example, the the zero crossing. Then, a_k and b_k will refer to the "in-phase" and "quadrature" spectral components, respectively.

In the more general case, the spectral coefficients can be computed over any window $[n, n+N]$, where the reference phase corresponding to sample n is $\phi[n] = \phi_0 \neq 0$. Then, a correcting rotation of $-k\phi_0$ can be applied to the complex DFT coefficient X_k:

$$X'_k = X_k \cdot e^{\phi_0 \cdot jk} \tag{4.12}$$

There are four common cases that require this phase rotation:

1. When calculating harmonic coefficients with a sliding window that is shifted by a non-integer number of periods, the start of each successive window will have a different phase ϕ_0, which must be accounted for by a rotation of X_k on a per-window basis.

2. For three-phase ac systems, it is common to use a single voltage $V_{\varphi A}$ as a phase reference. When computing spectral envelopes corresponding to $I_{\varphi A}$, $I_{\varphi B}$, and $I_{\varphi C}$ using this reference, phase rotations of 0°, 120°, and 240° should be applied, respectively.

3. Current transformers or transducers may introduce a fixed phase offset in their measurement, typically 0-1°. Correcting this offset with phase rotation of the preprocessor output can significantly reduce error at higher harmonics.

4. Multi-channel data acquisition cards that sample sequentially rather than simultaneously will introduce a similar phase offset between samples. For example,

evenly-spaced, 6 channel, 8 KHz sampling of a 60 Hz signal inserts a phase rotation of 0.45° between each channel.

In subsequent discussion, the first case is referred to as ϕ_{shift}, and the others are collectively referred to as ϕ_{extra}.

4.3 Spectral Envelope Preprocessor Implementation

Existing preprocessors such as the Kalman-filter spectral envelope preprocessor, described in [55], [54], and Section 4.4, have been heavily used in non-intrusive load monitoring and diagnostics [6–11,14–20,22,24,44,56,61]. However, a number of practical shortcomings of this preprocessor have been identified [62]. To address these, a new approach to spectral envelope preprocessing has been developed within the NilmDB framework, a unified system for managing and processing NILM data [62].

The new preprocessor has a modular, transparent design that allows for easy replacement and reuse of components, such as the phase alignment or harmonic coefficient calculation. It also fully supports the capabilities of NilmDB by utilizing its stream and metadata metaphors, and by incorporating accurate timestamping for all data. The preprocessor uses a phase and frequency estimation algorithm that is robust when presented with highly variable or truncated voltage waveforms. Finally, it focuses on correctness and implementation simplicity by taking advantage of the significant advancements in computing power since the existing preprocessors were developed.

Here, the spectral envelope preprocessor is split into two independent components. The first, an algorithm and code module called "`nilm-sinefit`", performs least-squares fits of sinusoids to successive windows of the input waveform in order to mark zero crossings, frequency, and amplitude. The second, "`nilm-prep`", performs spectral envelope extraction using (4.8), (4.9) and (4.12) over sliding windows of the input and sine fit data. The signal flow in the new spectral envelope preprocessor

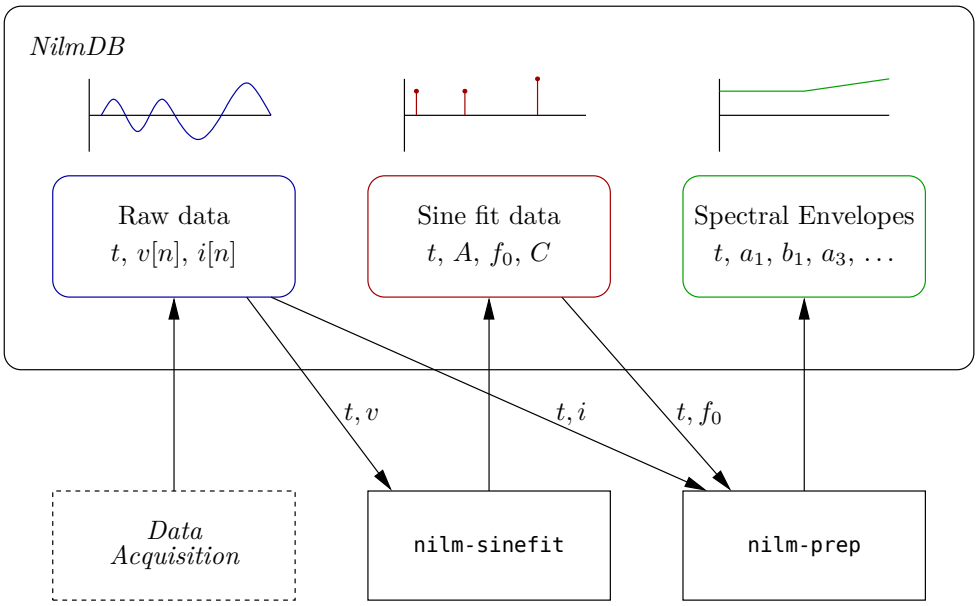


Figure 4-2: Block diagram of signal flow in the Sinefit spectral envelope preprocessor.

is shown in Figure 4-2, and the implementations of these components are detailed below.

4.3.1 4-Parameter Sinusoid Fitting

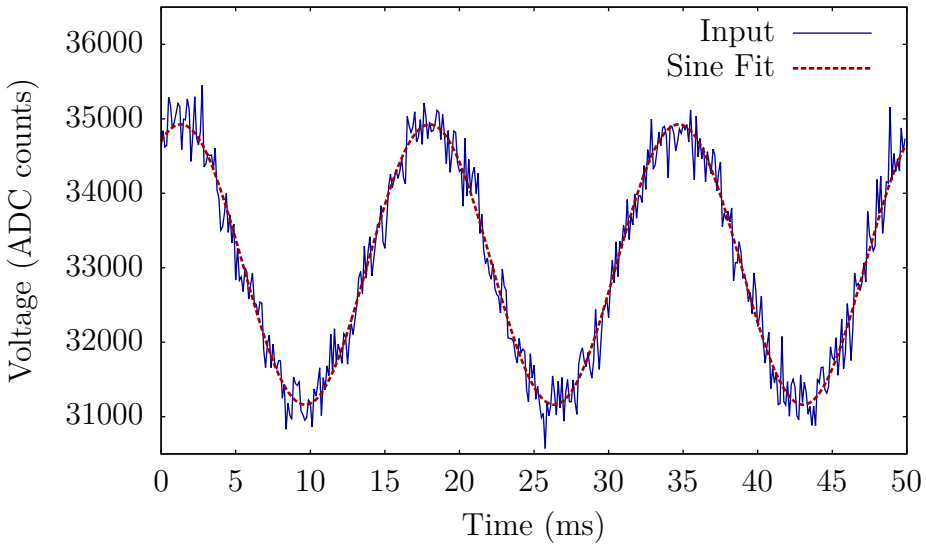
In order to accurately estimate the unknown frequency and phase angle from a voltage waveform, the preprocessor finds the best fit of a sinusoid to the data, using the method identified in IEEE Std 1241-2010 Annex B.2 [53]. Figure 4-3 demonstrates this optimal fit for two representative waveforms.[1]

Mathematically, given an arbitrary waveform vector $\mathbf{v}$ of length N sampled at frequency f_s, we wish to calculate the four parameters A', B', C, and f_0 that best satisfy:

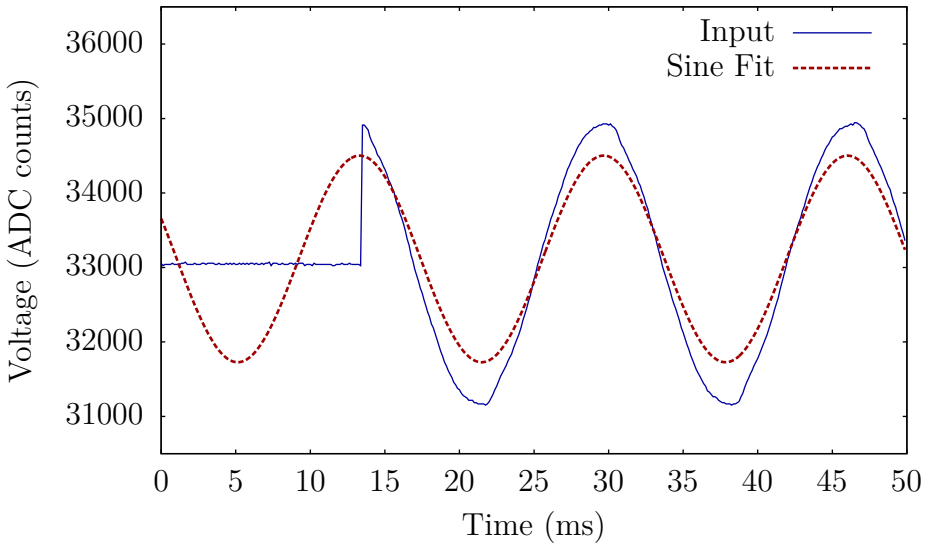
$$\mathbf{v}' = A' \cos\left(2\pi n \frac{f_0}{f_s}\right) + B' \sin\left(2\pi n \frac{f_0}{f_s}\right) + C \tag{4.13}$$

such that the least-squares residual $\sum(v[n] - v'[n])^2$ is minimized. Note that this system is non-linear.

[1]The modular design of the preprocessor allows other approaches to be easily used instead. For example, computationally restricted systems may find techniques such as the ones described in [63] to be more appropriate.



(a) Sine fit on a noisy waveform.



(b) Sine fit on a voltage that is zero for the first 1/4 of the window.

Figure 4-3: Results of 4-parameter sinusoid fitting on non-ideal waveforms. In both cases, frequency and phase are still identified with reasonable accuracy.

The algorithm that estimates these parameters is as follows. First, an initial estimate of f_0 is found by using an interpolated DFT [64,65]. This is done by computing $\mathcal{F}(\mathbf{v})$, the DFT of $\mathbf{v}$, and locating adjacent DFT indices L and $L+1$ corresponding to the maximum magnitude and its largest neighbor. The primary frequency component of the input lies between these two indices, and the estimate of its location L_{est} is calculated using the solution given by [66]. First, define U_1, V_1, U_2, and V_2 as the real and imaginary components of the adjacent DFT entries:

$$U_1 = \text{real}(\mathcal{F}(\mathbf{v})[L]) \tag{4.14}$$

$$V_1 = \text{imag}(\mathcal{F}(\mathbf{v})[L]) \tag{4.15}$$

$$U_2 = \text{real}(\mathcal{F}(\mathbf{v})[L+1]) \tag{4.16}$$

$$V_2 = \text{imag}(\mathcal{F}(\mathbf{v})[L+1]) \tag{4.17}$$

Then, the solution of the optimal intermediate index L_{est} is given by [66] as:

$$K_{opt} = \frac{(V_2 - V_1)\sin(\frac{2\pi}{N}L) + (U_2 + U_1)\cos(\frac{2\pi}{N}L)}{U_2 - U_1} \tag{4.18}$$

$$Z_1 = V_1\left(\frac{K_{opt} - \cos(\frac{2\pi}{N}L)}{\sin(\frac{2\pi}{N}L)}\right) + U_1 \tag{4.19}$$

$$Z_2 = V_2\left(\frac{K_{opt} - \cos(\frac{2\pi}{N}(L+1))}{\sin(\frac{2\pi}{N}(L+1))}\right) + U_2 \tag{4.20}$$

$$L_{\text{est}} = \arccos\left(\frac{Z_2\cos(\frac{2\pi}{N}(L+1)) - Z_1\cos(\frac{2\pi}{N}L)}{\frac{2\pi}{N}(Z_2 - Z_1)}\right) \tag{4.21}$$

The initial frequency estimate corresponding to L_{est} is:

$$f_{0,\text{est}} = L_{\text{est}} \cdot f_s/N \tag{4.22}$$

Now, perform the iterative fit of (4.13). Define the starting conditions as:

$$A_0 = B_0 = C_0 = 0 \tag{4.23}$$

$$\omega_0 = 2\pi \cdot f_{0,\text{est}} \tag{4.24}$$

For each iteration $i = \{0, 1, 2, \ldots, m\}$, perform a least-squares fit on a linearization of the system. Given an assumed small deviation from ω_i of $\Delta\omega_i$, the Taylor series expansion of (4.13) around the current estimated parameters gives the linear equation:

$$\mathbf{v_i'} = \mathbf{D_i} \cdot \begin{bmatrix} A_i & B_i & C_i & \Delta\omega_i \end{bmatrix}^T \tag{4.25}$$

where matrix $\mathbf{D_i}$ is given by:

$$t_i = (i-1)/f_s \tag{4.26}$$

$$\mathbf{D_i} = \begin{bmatrix} \cos(\omega_i t_1) & \sin(\omega_i t_1) & 1 & B_i t_1 \cos(\omega_i t_1) - A_i t_1 \sin(\omega_i t_1) \\ \cos(\omega_i t_2) & \sin(\omega_i t_2) & 1 & B_i t_2 \cos(\omega_i t_2) - A_i t_2 \sin(\omega_i t_2) \\ \vdots & \vdots & \vdots & \vdots \\ \cos(\omega_i t_N) & \sin(\omega_i t_N) & 1 & B_i t_N \cos(\omega_i t_N) - A_i t_N \sin(\omega_i t_N) \end{bmatrix} \tag{4.27}$$

The least-squares solution to this system gives the updated estimates for the next iteration as:

$$\begin{bmatrix} A_{i+1} \\ B_{i+1} \\ C_{i+1} \\ \Delta\omega_{i+1} \end{bmatrix} = \left(\mathbf{D_i}^T \mathbf{D_i}\right)^{-1} \left(\mathbf{D_i}^T \mathbf{v}\right) \tag{4.28}$$

$$\omega_{i+1} = \omega_i + \Delta\omega_{i+1} \tag{4.29}$$

Note that the least-squares fit in (4.28) can be computed with a more numerically-stable method such as Q-R decomposition [53]. The solution converges rapidly, and the preprocessor stops after a fixed number $m = 7$ iterations. The fitted parameters for (4.13) are:

$$A' = A_m \qquad B' = B_m \tag{4.30}$$

$$C = C_m \qquad f_0 = \frac{\omega_m}{2\pi} \tag{4.31}$$

Finally, we convert this fit into the equivalent polar form:

$$v[n] = A \cdot \sin\left(2\pi n \frac{f_0}{f_s} + \phi_0\right) + C \tag{4.32}$$

by computing:

$$A = \sqrt{A_m^2 + B_m^2} \tag{4.33}$$

$$f_0 = \frac{\omega_m}{2\pi} \tag{4.34}$$

$$\phi_0 = \text{atan2}(A_m, B_m) \tag{4.35}$$

$$C = C_m \tag{4.36}$$

This form is preferable, because the parameters $[A, f_0, \phi_0, C]$ are more directly applicable to computing spectral envelopes.

4.3.2 Implementation of `nilm-sinefit`

The `nilm-sinefit` tool, implemented in Python, uses successive 4-parameter sine wave fits to find and mark every positive zero crossing ($\phi = 0$) of an input voltage waveform. Each mark includes the amplitude A, frequency f_0, and offset C of the following period. Given the expected line frequency f_{exp} and sampling rate f_s, the fits are calculated over sliding windows of

$$N = 3.5 \frac{f_s}{f_{\text{exp}}} \tag{4.37}$$

samples of voltage. This corresponds to approximately 3.5 periods, although the actual number will vary with f_0. If the fitted f_0 falls outside predetermined bounds, or the amplitude A is too low, the window is skipped to avoid spurious marks.

Figure 4-4 demonstrates the fit and marking over a window of length N. To avoid potentially double-marking a zero crossing that occurs near window boundaries, the point N_s is calculated as the last point within $[0, N]$ with phase angle $\frac{\pi}{2}$. Only zero crossings prior to N_s are marked, and the next window is shifted to $[N_s, N_s + N]$. This

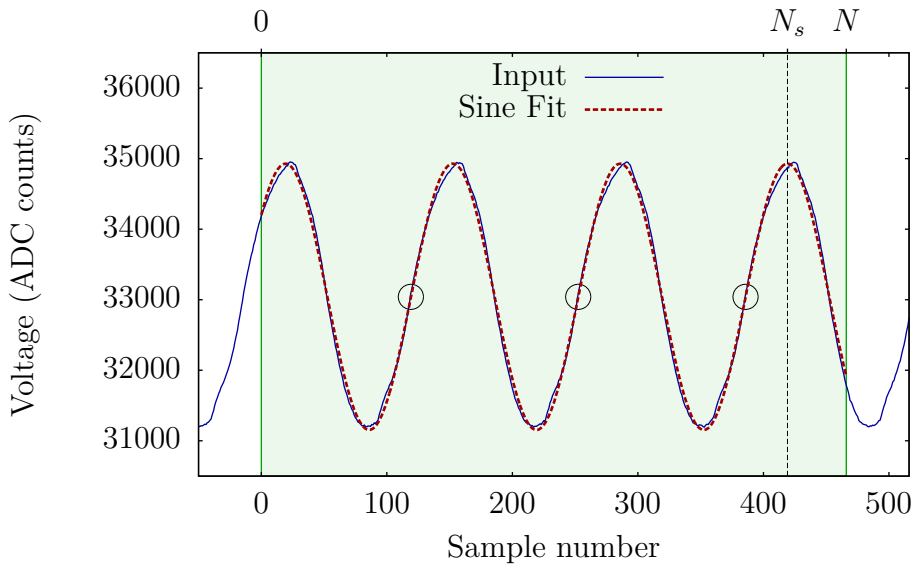


Figure 4-4: Windowing algorithm of `nilm-sinefit`. Sine fit is performed over the window $[0, N]$, and positive zero crossings are marked as shown by the circles. The next window starts at N_s, the last point with phase angle $\frac{\pi}{2}$.

ensures that any particular zero crossing will only fall squarely within one window, even as sine fit parameters change.

The `nilm-sinefit` processing tool takes as input a single NilmDB stream, and marks zero crossings using data from a user-specified column in the input stream. Output marks are written to a new stream consisting of a timestamp t and three floating-point values A, f_0, and C. These values correspond to the parameters from (4.32), as calculated for the crossing detected at time t.

The source code for `nilm-sinefit` can be found in Listing F-1, and a command-line reference can be found in Section 3.4.2.10.

4.3.3 Implementation of `nilm-prep`

The `nilm-prep` tool, implemented in Python, reads two data streams: an input waveform $i[n]$, and the zero crossing and f_0 estimates for each period as marked by `nilm-sinefit`. Other parameters that control its behavior are shown in Table 4-1. For each identified zero crossing at time t with frequency f_0, the spectral envelopes are calculated with the following algorithm:

Parameter	Default	Description
N_{harm}	4	Number of odd harmonics to store.
N_{shift}	1	Number of shifted windows for which to compute coefficients, per zero crossing.
ϕ_{extra}	0	Extra phase rotation to apply, to correct for known phase offsets.

Table 4-1: Configurable parameters for `nilm-prep`.

1. Initialize $N = f_s/f_0$ and $\phi_{\text{shift}} = 0$.
2. Extract N samples starting at time $t + (\phi_{\text{shift}}/(2\pi f_0))$.
3. Use the Fast Fourier Transform to calculate X_k of $i[n]$, as defined in (4.6).
4. Apply rotation from (4.12) using $\phi_0 = \phi_{\text{extra}} - \phi_{\text{shift}}$ to obtain X'_k.
5. Calculate and store the first N_{harm} odd harmonic coefficients from X'_k using (4.8) and (4.9).
6. Increment ϕ_{shift} by $2\pi/N_{\text{shift}}$.
7. Repeat from step 2 until $\phi_{\text{shift}} \geq 2\pi$.

This results in N_{shift} sets of coefficients per period of the input waveform, by applying successive overlapping DFTs. This sliding-window approach may be useful in some cases, as it can help retain additional information about energy content at harmonics that are not otherwise stored by this implementation, such as the even harmonics. For waveforms known to consist primarily of the low-numbered odd harmonics, $N_{\text{shift}} = 1$ is sufficient and results in the most space savings.

Output from `nilm-prep` is stored in a new NilmDB stream where each row contains a timestamp and $2 * N_{\text{harm}}$ floating-point values numbers, in order $\{a_1, b_1, a_3, b_3, \ldots\}$.

The source code for `nilm-prep` can be found in Listing F-2, and a command-line reference can be found in Section 3.4.2.9.

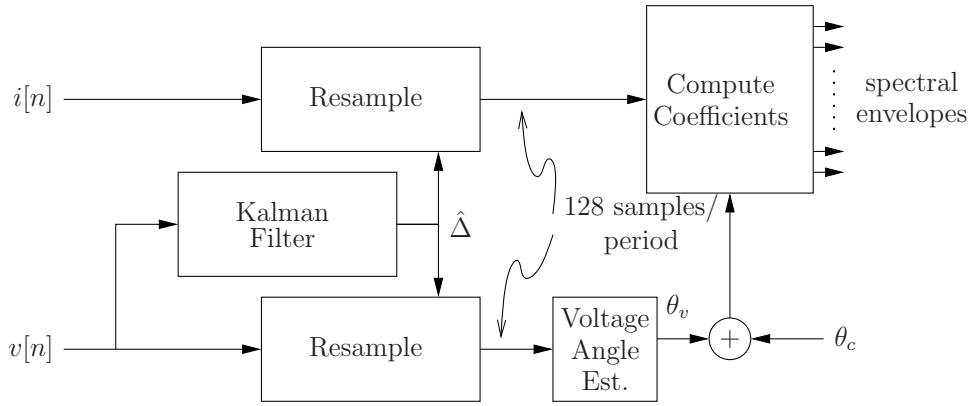


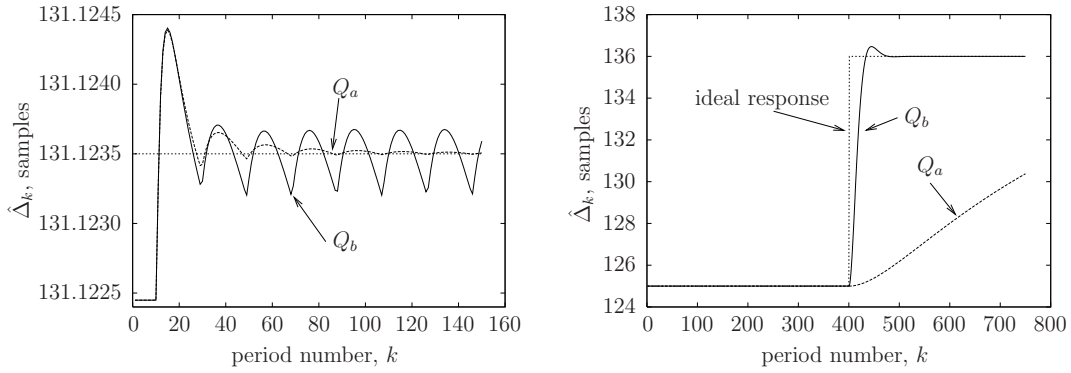
Figure 4-5: Block diagram of the Kalman-filter spectral envelope preprocessor, from [54].

4.4 Comparison with Existing Preprocessors

4.4.1 Kalman-filter Preprocessor

The Kalman-filter spectral envelope preprocessor, described in [55] and [54], has been heavily used in non-intrusive load monitoring and diagnostics [6–11, 14–20, 22, 24, 44, 56, 61]. A block diagram of this preprocessor is shown in Figure 4-5. It is written in C, and is passed two vectors of single-phase input data, $v[n]$ and $i[n]$. Positive zero crossings of $v[n]$ are detected using a noise-tolerant thresholding, and these zero crossings are fed into a discrete Kalman filter which extracts the fundamental frequency f_0.

The input data $i[n]$ is resampled so that there are exactly 128 points per input period. After this, a straightforward computation of (4.4) is performed by multiplying a one-period window of the resampled data with precomputed "basis vectors" consisting of the in-phase and quadrature harmonics $\sin(kt)$ and $\cos(kt)$ for $k = 1, 3, 5, 7$. This basis vector multiplication is an optimization over using the FFT to compute (4.6) directly, when only a few of the outputs are required. Rotations are then applied to the harmonic coefficients to account for the phase of the start of the window, and for inter-sample phase offset between the $i[n]$ and $v[n]$. The window is then moved by one half-period, resulting in an output of two coefficients per input period (120 Hz



(a) Lock-in performance

(b) Step response

Figure 4-6: Lock-in performance and step response of the Kalman filter as a function of Kalman covariance matrix Q, from [54].

for a typical ac system).

While this preprocessor has been successfully used for many load monitors, a number of drawbacks have become apparent. The primary issue has been that the zero crossing detection and Kalman filter can be sensitive to noise and transient disruptions in the voltage signal, and is not particularly suitable for variable frequency drives with widely varying frequencies. Figures 4-6a and 4-6b demonstrate the Kalman filter's ability to lock to a signal and respond to a step change in frequency, which is controlled by parameter Q. Low Q is necessary to avoid ripple in the frequency estimate, but can cause the filter to take hundreds to thousands of line cycles in order to track a change in frequency. Even with high Q, tens or hundreds of cycles are not uncommon. This leads to a large asymmetry between the filter's response and the respective preprocessor output before and after the frequency change, which can negatively affect some diagnostics.

A related issue is that the Kalman filter relies on an existing state estimate, and can lose stability in situations where that state estimate is too large. One common real-world installation issue is that the voltage input will occasionally become temporarily disconnected, due to operator mistake, equipment failure, or power outage. In such cases, we observed that the Kalman filter would sometimes never recover lock, depending on its random state evolution due to measured noise during the signal loss. The "menu-system" data collection software developed in [22] was designed to force-

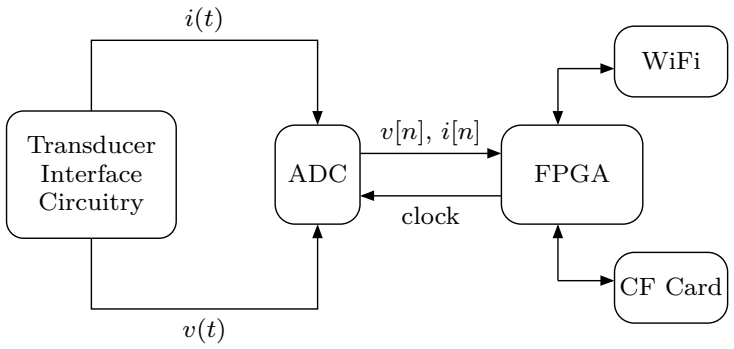


Figure 4-7: FPGA-based spectral envelope preprocessor, block diagram, from [12].

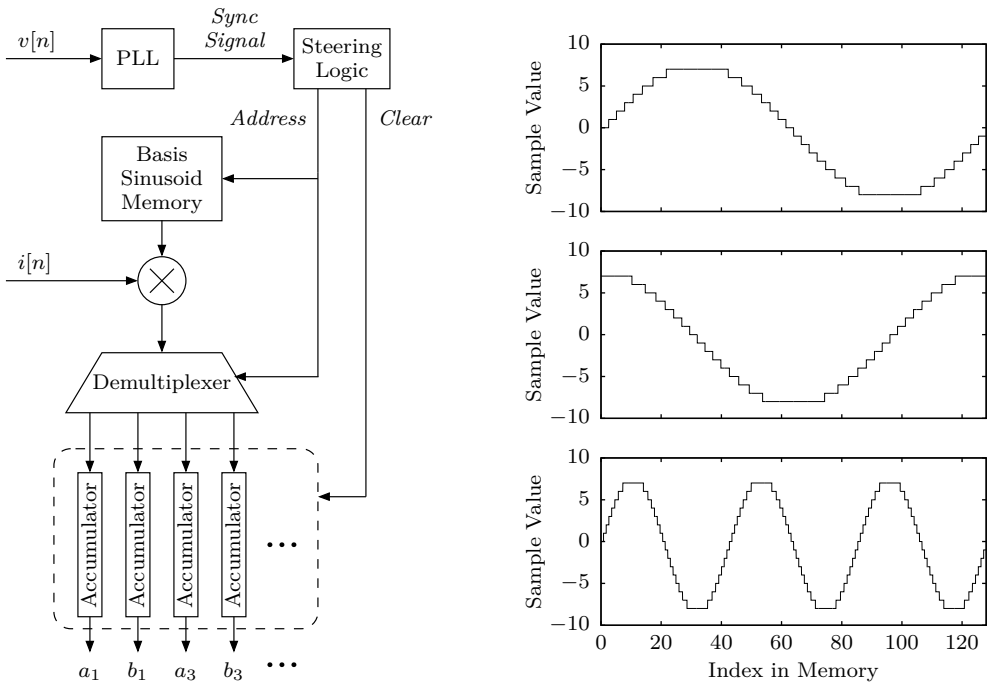
fully close and restart the preprocessor several times a day to mitigate this issue, at the expense of creating discontinuities in the recorded data.

Finally, the monolithic C implementation of the Kalman-filter spectral envelope preprocessor makes it difficult to directly correlate input with output. The output stream provides only the harmonic coefficients, with neither the extracted line frequencies nor any timestamps that can be used to line up harmonics with a particular range of input samples. This is a minor inconvenience when dealing with a stiff utility frequency that can be assumed constant over a long period of time, but can lead to significant difficulty in tracking event timing when dealing with variable or unstable drives.

4.4.2 FPGA-based Preprocessor

A field-programmable gate array (FPGA)-based data acquisition, preprocessor, storage, and wireless transmission system was developed in [12]. The goal of this implementation was to demonstrate an integrated, low-power, hardware-based approach to spectral envelope measurement. A system block diagram and an image of the prototype are shown in Figure 4-7. A detailed look at the implementation of the preprocessor within the FPGA is shown in Figure 4-8.

Like the Kalman-filter preprocessor, the FPGA-based preprocessor utilizes precomputed basis sinusoid vectors with which to multiply the raw $i[n]$ waveform in order to extract harmonic coefficients. The primary difference is in the phase alignment and detection of the utility frequency f_0. Whereas the Kalman filter extracted



(a) Signal flow (b) Three of the eight basis sinusoids

Figure 4-8: FPGA-based spectral envelope preprocessor design, from [12].

this information post-sampling and then resampled to create 128 data points per period, the FPGA system utilizes a variable sampling rate ADC under real-time control. The sampling rate f_s is adjusted by an on-board PLL and steering logic to ensure a sampling rate of $f_s = 128 \cdot f_0$.

The linear resampling was the "most significant source of error" in the Kalman-filter preprocessor [54]. By using a variable sampling rate instead, the FPGA preprocessor avoids this error while also avoiding the excessive resource utilization that would be required by a more accurate resampling filter. However, it has a significant practical drawback in that it must be performed on-line; that is, it cannot easily be used to post-process existing sampled data.

The hardware-centric nature of the FPGA system also precludes easy modification to add potentially needed features, such as the ability to process three-phase data, apply arbitrary rotations, store raw samples alongside the spectral envelopes, or add timestamps to the output stream. For low power and other constrained environments, though, it serves as a useful example of how appropriate hardware can be used to

efficiently "skip" directly to spectral envelopes when storing or transmitting data.

4.5 Compression Benefits

Physical modeling of ac loads shows that there are often useful bounds on the number and types of harmonics that are present in the load current. By omitting these higher harmonics from stored data streams, the spectral envelope preprocessor performs a domain-specific form of compression that greatly reduces data storage and transfer requirements. A typical NILM data acquisition is 16-bit samples at a rate of 8 KHz. For a three-phase system, 6 channels are recorded. Assuming one 64-bit timestamp is also stored with each sample, the storage requirements for raw data are:

$$\begin{aligned} 8 \text{ KHz} \cdot (64 + 16 \cdot 6) \text{ bits} &= 1{,}280{,}000 \text{ bits/s} \\ &= 160 \text{ kB / s} \\ &= 576 \text{ MB / hour} \\ &= 13.8 \text{ GB / day.} \end{aligned} \tag{4.38}$$

Preprocessor output is harmonics a_k and b_k for $k = 1, 3, 5, 7$ as 32-bit floating point values with a 64-bit timestamp, for each of the three phases. One set of coefficients is calculated per 60 Hz period, so the total storage requirements for preprocessed data are:

$$\begin{aligned} 60 \text{ Hz} \cdot (64 + 32 \cdot 8) \text{ bits} \cdot 3 &= 57{,}600 \text{ bits/s} \\ &= 7.2 \text{ kB / s} \\ &= 25.9 \text{ MB / hour} \\ &= 0.62 \text{ GB / day.} \end{aligned} \tag{4.39}$$

Thus, the preprocessor reduces the data to only 4.5% of its original size, while preserving all information about three-phase power usage up to the 7th harmonic. Even doubling this and storing up to the 15th harmonic still uses only 8.1% of the original

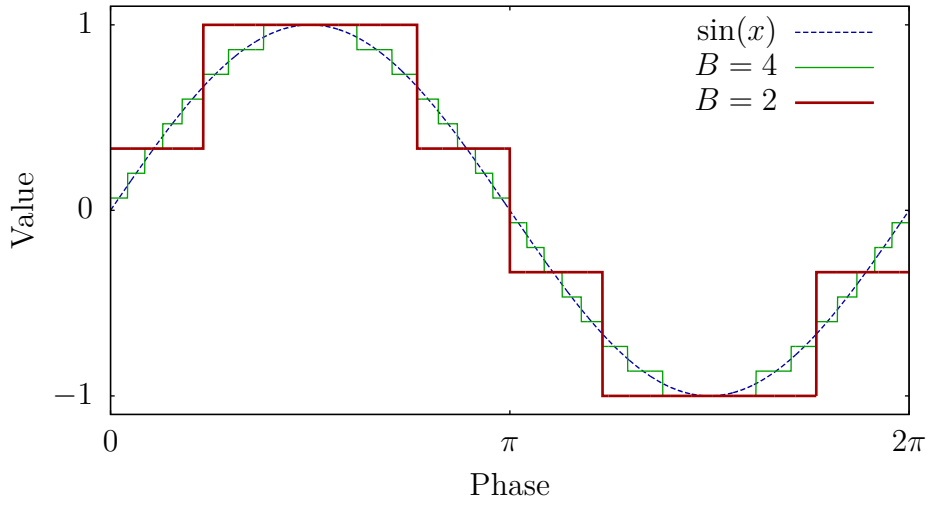


Figure 4-9: Quantization error, as would be introduced by a linear ADC. The continuous function $\sin(x)$ is compared to the same function represented discretely with only 4 and 2 bits of resolution.

raw data space.

Other reductions of the data could instead be applied, for example, simply recording aggregate real power averaged over every period or second. However, such data would not reflect the detailed short term variations that would occur in real power, nor would it reflect any of the behavior of the higher harmonics. Time-varying spectral envelope coefficients strike a balance between the usefulness of the retained data and the storage and transmission requirements of that data.

4.6 Resolution and Accuracy

The original digital samples of the current and voltage waveforms have limited precision due to quantization. As shown in Figure 4-9, the continuous input signal is divided into B discrete regions, and each region is mapped to a unique digital code during the sampling process. This quantization can be stated explicitly as

$$i[n] = \left\lfloor 2^B \cdot i(t) + \frac{1}{2} \right\rfloor \tag{4.40}$$

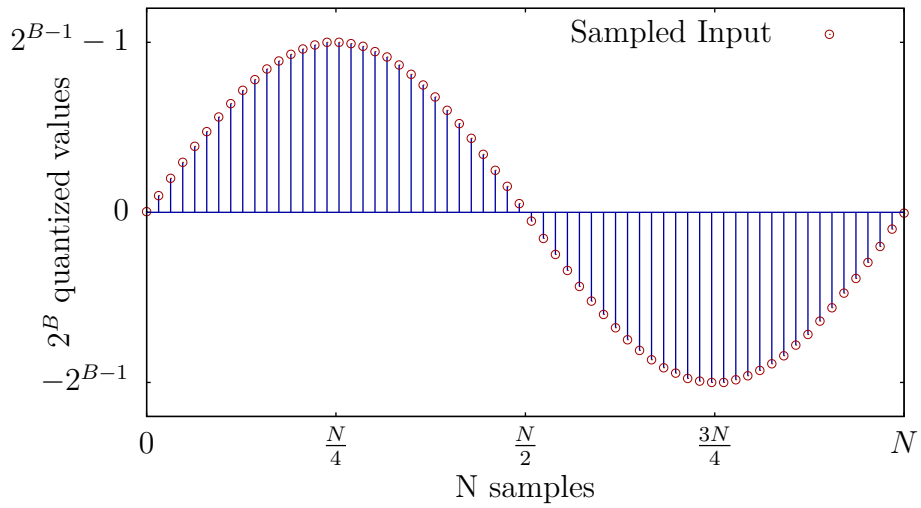


Figure 4-10: Sampled waveform data over one line period. These N samples at B bits of resolution comprise the input to the spectral envelope preprocessor.

where $i(t)$ is normalized to the range $[0, 1]$. The preprocessor input consists of N values of $i[n]$ over one period of the line voltage. Figure 4-10 shows this input as sampled from a pure sinusoid at the fundamental frequency.

For power estimation and load identification purposes, it is desirable to find the average power of the input signal, but the quantization error affects this measurement. Using raw data to directly estimate power is less accurate than using preprocessed spectral envelopes to do the same. To compare the accuracy, we begin by considering estimation of power when the current waveform is a simple sinusoid of the form

$$i(t) = A\sin(\omega t) \tag{4.41}$$

Here, total power is proportional to the amplitude A of the sinusoid. More complex input waveforms are addressed in Section 4.6.5.

4.6.1 Resolution of Power From Raw Data

A straightforward approach to estimating A from raw sampled data is to locate the peak of the waveform. However, as shown in Figure 4-11, finding the amplitude in this manner leads to measurement error, because the single point at the peak is subject

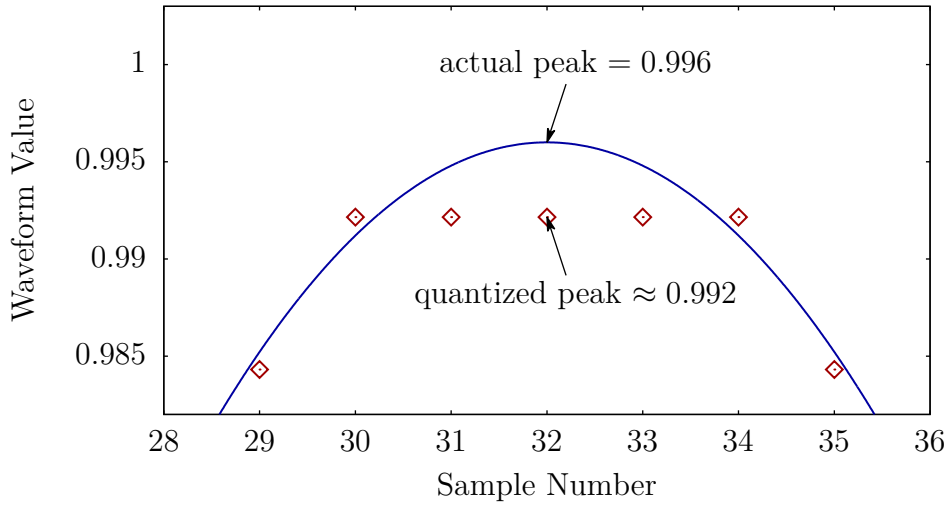


Figure 4-11: Example of the error caused by estimating signal amplitude using the peak quantized sample value alone. Here, $A = 0.996$, $N = 128$, $B = 8$.

to the same quantization as any other point. Of the 2^B possible quantized sample values, the positive peak will reside in the upper half, resulting in a total of $2^{(B-1)}$ discernable amplitudes. This corresponds to an overall resolution of

$$\log_2 2^{(B-1)} = B - 1 \tag{4.42}$$

bits of precision, slightly less than the original sampling precision of B bits.

4.6.2 Resolution of Power From Preprocessor Output

The preprocessor improves effective power reesolution by averaging over time. Specifically, `nilm-prep` uses not only the quantized value at the peak, but all N sampled data points over one or more periods. For example, the preprocessor discards all but the low, odd-numbered harmonics; the sinusoid in (4.41) contains no harmonics other than for $k = 1$. Thus, the `nilm-prep` output encapsulates all information about the original signal. Since the analogous DFT is invertible, this means that every unique sampled input has a unique corresponding preprocessor output.

We can use this to determine how many discernible outputs the preprocessor will produce as the amplitude A of the input sinusoid changes. The sampling process that

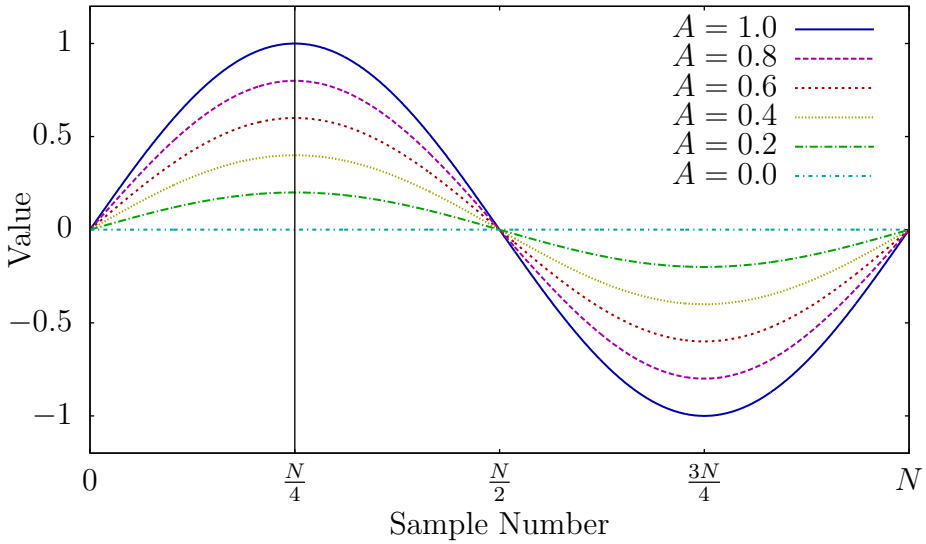


Figure 4-12: Sweeping through all input amplitudes of a single fundamental frequency from 0 to 1, in order to enumerate the number of possible quantized inputs. By symmetry, only the first $N/4$ samples need to be considered.

generates $i[n]$ is limited in resolution, and so there are necessarily a finite number U of unique sampled $i[n]$ waveforms. Because of the 1:1 relationship of the DFT, there are the same number U of unique preprocessor outputs, and we can therefore discern U different values of A.

To calculate U, consider sweeping A from $0 \rightarrow 1$, as demonstrated in Figure 4-12. By symmetry, let us consider only the first $N/4$ samples. At $A = 0$, all samples are zero. As A increases, there will be some transition where a single quantized sample $i[n]$ will increase by one. More specifically, as A increases from 0 to 1, samples will monotonically increase according to the following pattern:

- The first sample at $n = 0$ never increases.
- The peak sample at $n = N/4$ increases $2^{(B-1)} - 1$ times.
- For each sample between these extremes, the number of "increases", or effective quantization steps, is given by:

$$U_n = \left\lfloor (2^{(B-1)} - 1) \sin\left(\frac{2\pi n}{N}\right) + \frac{1}{2} \right\rfloor \tag{4.43}$$

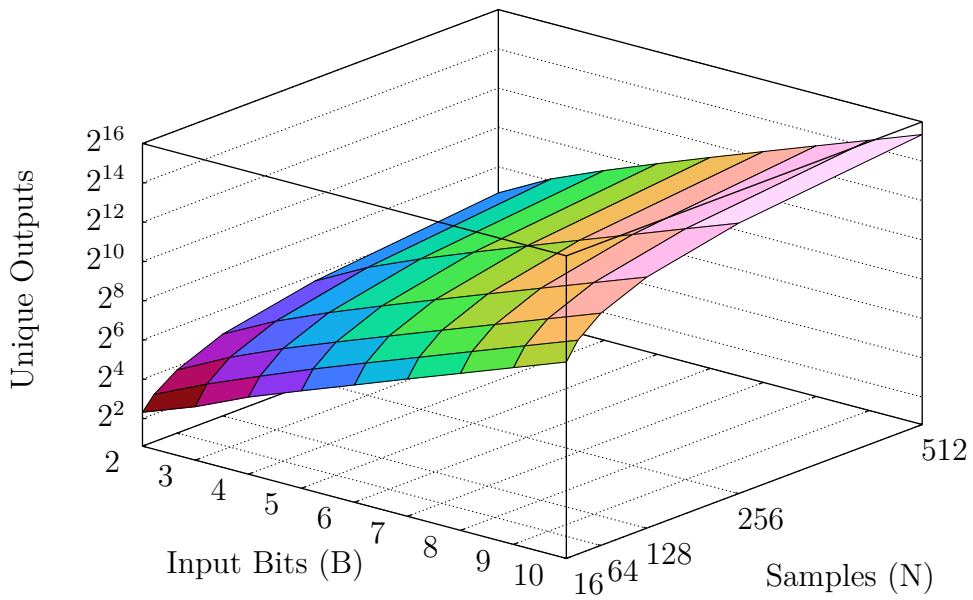


Figure 4-13: Number of unique preprocessor outputs, as a function of the number of samples N and the number of input bits B, for an input signal consisting of a single sinusoid of varying amplitude.

where $\lfloor x \rfloor$ denotes floor(x).

Each increase creates a new input to the preprocessor, which results in a new output. Thus, the total number of unique outputs from the preprocessor is one corresponding to the initial case ($i[n] = 0$), plus one for each time a sample in $i[n]$ increases.

$$U = 1 + 0 + \left(\sum_{n=1}^{\frac{N}{4}-1} U_n \right) + \left(2^{(B-1)} - 1 \right) \tag{4.44}$$

$$= 2^{(B-1)} + \sum_{n=1}^{\frac{N}{4}-1} \left\lfloor (2^{(B-1)} - 1) \sin\left(\frac{2\pi n}{N} \right) + \frac{1}{2} \right\rfloor \tag{4.45}$$

Figure 4-13 shows this number of unique outputs U as a function of the input samples N and input quantizer bits B. The number of output bits are approximately linear with the number of input bits, and increase logarithmically with N. As a representative data point, sampling at values $N = 128$ and $B = 10$ gives an overall output resolution of about $B = 13$ bits.

Note, however, that the unique outputs of the preprocessor are not linearly dis-

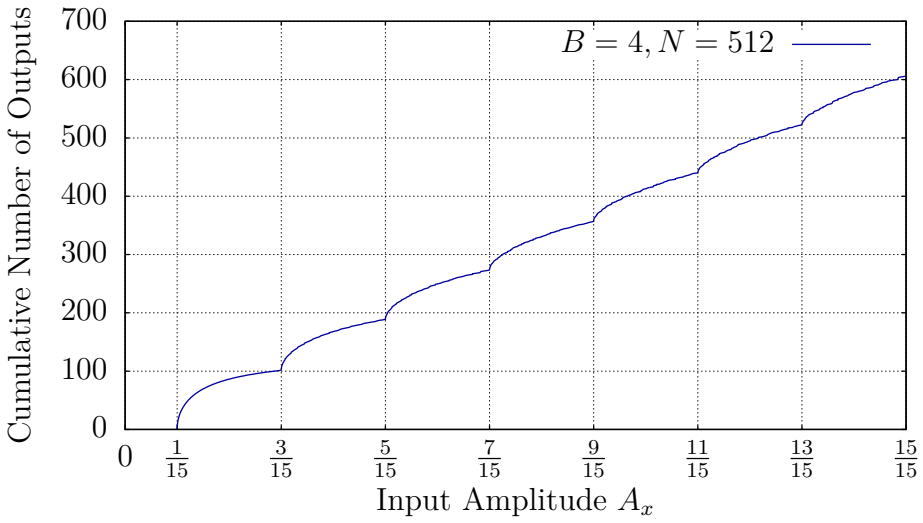


Figure 4-14: Unique preprocessor outputs as a function of input amplitude of a sinusoid. For each amplitude level A_x, the graph plots the total number of unique outputs seen while the input sinusoid amplitude is swept from 0 to A_x.

tributed among the input amplitudes. For example, for amplitudes less than the first quantized bit level of $1/(2^B - 1)$, all quantized input samples are zero, and so there is only one unique preprocessor output. Figure 4-14 shows the number of unique outputs as a function of the input amplitude, as enumerated computationally via binary search (described later in Algorithm 4-3). As the input amplitude is swept along the x-axis from left to right, the cumulative number of unique outputs are counted and plotted along the y-axis. Inflection points in the curve correspond to the quantization levels. This is because small changes in amplitude affect more samples when the peak of the sinusoid is near a quantization level, causing relatively more unique outputs in these areas.

From Figure 4-14, a measure of effective resolution as a function of input amplitude can be developed. If a change of amplitude from A to $(A + \Delta A)$ causes the preprocessor to output a new unique value, then discerning these values corresponds to discriminating between input amplitudes with a resolution of $\beta = \log_2(1/\Delta A)$ bits around operating point A. Figure 4-15 shows this resolution as a function of input amplitude. Here, the input bits $B = 4$ and samples $N = 512$, and the output resolution varies between $\beta = 4$ and 18 bits, with an average around 10 bits.

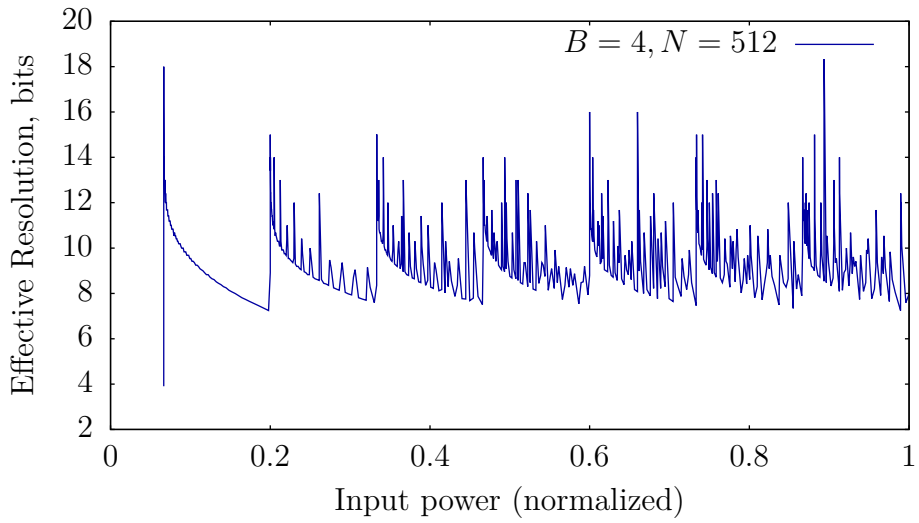


Figure 4-15: Effective resolution, in bits, as a function of input amplitude of a sinusoid. An effective resolution of β bits means that a change of $1/(2^{\beta})$ in input amplitude can be resolved.

4.6.3 Accuracy of Preprocessor Output

For load identification, the ability to simply discern inputs may be sufficient. As shown, the preprocessor can provide a high resolution estimate of input waveform amplitude. In some cases, such as in power level measurement and energy scorekeeping, high accuracy is also desired.

When the characteristics of the input waveform are known, the accuracy can often be directly determined. To continue the previous example, consider a single sinusoidal waveform at the fundamental line frequency and amplitude A, corresponding to power P. This waveform is sampled, quantized, and passed through the spectral envelope preprocessor. Figure 4-16 shows a plot of the error between the preprocessor output P_1 and the actual power P, as a function of P. Like the resolution, the error varies with signal amplitude. Note that the maximum absolute error corresponds to approximately one part in 2^B, where B is the quantization bits. Thus, the plot shows that the error of the preprocessor output is always less than or equal to the quantization error of any individual sample.

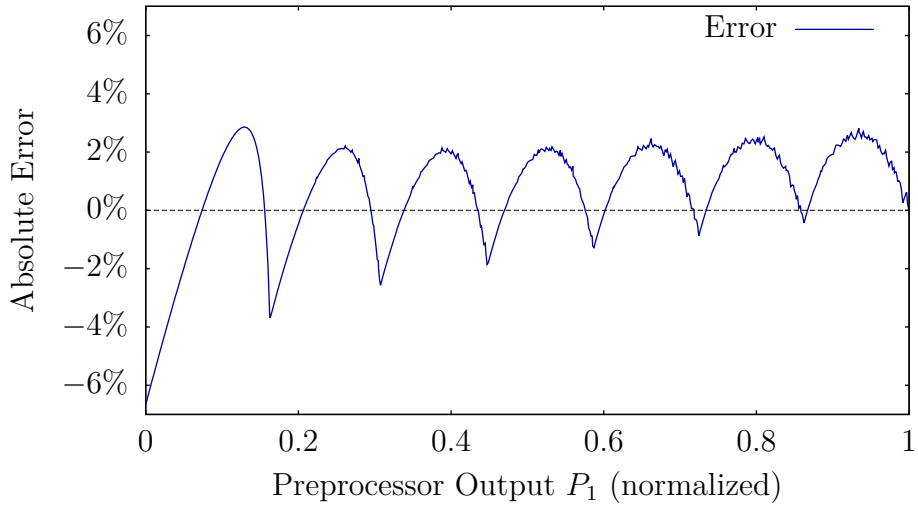


Figure 4-16: Accuracy of preprocessor output when provided with quantized input data from a single sinusoid. The absolute error, as a percentage of the full power range, is plotted as a function of the P_1 output. Here, $B = 4, N = 512$, and the maximum error of any single sample due to quantization alone would be $1/(2^B) = 6.25\%$.

4.6.4 Accuracy in the Presence of Noise

The previous analysis has been calculated and simulated based on ideal waveforms. In practice, there are many potential sources of noise or interference in non-intrusive load monitoring. This noise can be broadly split into two categories, correlated and uncorrelated signals.

Correlated interference refers to any introduced signal or distortion that is related to the loads being monitored or is otherwise nonrandom. Examples include magnetic coupling between adjacent current transducers, pickup of stray 60 Hz electric fields from other shielded wiring or lighting fixtures, and aliasing effects in the data acquisition process. This sort of coupling can potentially be complex and relate closely to specific installation details, which means that its effect on preprocessor accuracy is highly variable. In general, the net effect of correlated noise has been extremely low in observed systems [13].

Uncorrelated noise is statistically independent from the input waveforms. One such type of noise is additive white Gaussian noise (AWGN), which is normally-distributed around zero and might be expected to appear as a result of thermal noise

in the sensors or data acquisition. This form of noise often sets the limit of sampling resolution in the monitoring system.

The preprocessor is particularly well-suited to handle such disturbances. To analyze the effect of AWGN on preprocessor accuracy, we again consider the example of a single sinusoid, with added noise $\mathcal{N}(t)$:

$$i(t) = A\sin(\omega t) + \mathcal{N}(t) \tag{4.46}$$

By sweeping A from $0 \rightarrow 1$, we can generate data in the same manner as Figure 4-16, simulating the sampling process and calculating the preprocessor power estimate $P_1(A)$ corresponding to each actual power input A. Define the overall root-mean-square error of this data as:

$$E_{\text{RMS}} = \sqrt{\int_0^1 \left(P_1(A) - A\right)^2 dA} \tag{4.47}$$

This RMS error will vary with the amount of noise $\mathcal{N}(t)$ that is being added to the sinusoid. We describe this amount using the signal-to-noise ratio (SNR), the ratio of power in the full-amplitude sinusoid to the power of the added white Gaussian noise. Note that the amount of noise is held constant as A is swept.

Figure 4-17 shows the calculated overall RMS error versus the SNR of injected noise. Two approaches to estimating power are shown: the preprocessor P_1 estimate, and the simple peak-estimation technique described in Section 4.6.1. As expected, the preprocessor estimate offers an improvement over the raw peak estimate in all cases, reducing overall error by more than half.

In some cases, the addition of noise will actually *reduce* error. Here, noise levels around 30 dB provide a slight accuracy improvement. This is due the dithering effect of the white Gaussian noise on the sampling quantization which, combined with the averaging effect of the spectral envelope calculation, serves to decouple the quantization error from the input waveform [67–69]. The preprocessor can therefore be seen as even more useful in the presence of noise.

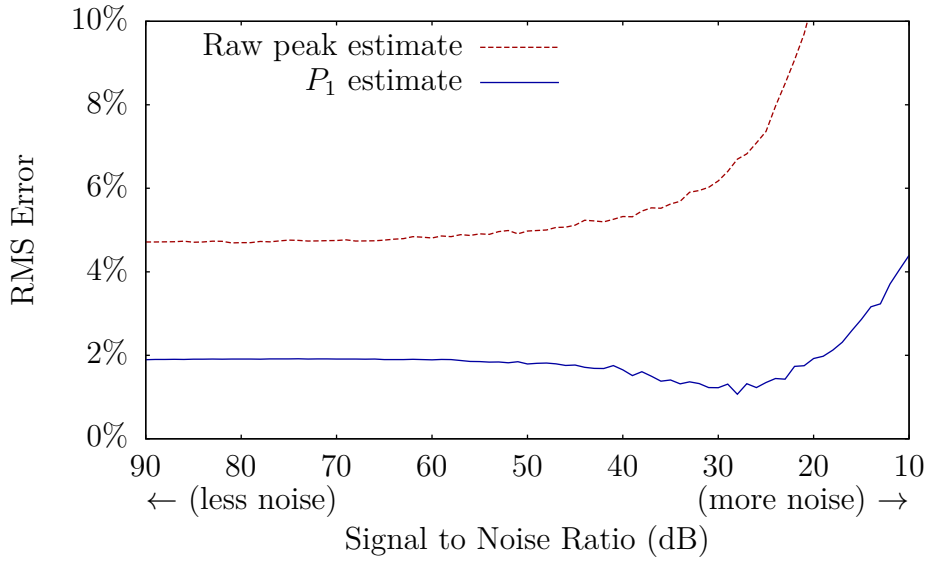


Figure 4-17: Total RMS error of power estimates for a single sinusoid over all amplitudes $A = 0 \rightarrow 1$, with simulated additive Gaussian white noise injected at the specified signal-to-noise ratio. At some levels, white noise can improve preprocessor accuracy by introducing dither into the quantization process. The sampling parameters are $B = 4, N = 512$.

4.6.5 Resolution and Accuracy with Complex Waveforms

Real-world systems often draw energy content at more harmonics than just the fundamental line frequency. The analysis of preprocessor resolution and accuracy can be extended past single sinusoids to take these harmonics into account. For such waveforms, closed-form solutions to find the number of discernable power levels like (4.45) may not necessarily exist.

Instead, we develop a model $f(A)$ that describes the current waveform shape as a function of power level $A = 0 \rightarrow 1$. This approach is suitable for loads that exhibit correlations between fundamental and higher harmonic components, such as a_3 and a_5. For example, one such model is described in [70] for variable-speed drive (VSD) systems, where harmonic contents are related to the total apparent power A by functions of the form:

$$\sqrt{a_k^2 + b_k^2} = bA^p \tag{4.48}$$

```
outputs ← [ ]
function ENUMERATE(f, A_low, A_high)
    P_low ← PREPROCESS(SAMPLE(f(A_low)))
    P_high ← PREPROCESS(SAMPLE(f(A_high)))
    if P_low ≠ P_high then                          ▷ More than one output in this interval?
        A_mid ← (A_high + A_low)/2
        ENUMERATE(f, A_low, A_mid)                  ▷ Search 1st half
        ENUMERATE(f, A_mid, A_high)                 ▷ Search 2nd half
    else if P_low not in outputs then
        outputs ← outputs + P_low                   ▷ One new unique output; store it
ENUMERATE(f, 0, 1)
return SORT(outputs)
```

Algorithm 4-3: Binary search to enumerate unique preprocessor outputs as power A varies, given waveform model $f(A)$.

In many systems, including VSDs, these relationships are nearly linear. Here, we consider a similar class of waveforms where the ratios between harmonics are fixed in proportions A_3 and A_5:

$$f(A,t) = A \cdot (\sin(\omega t) + A_3 \sin(3\omega t) + A_5 \sin(5\omega t)) \tag{4.49}$$

We wish to determine the effective resolution of the preprocessor for systems following this model as we vary A. As with the single sinusoid in Section 4.6.2, we do this by counting the number of unique preprocessor outputs as we sweep $A = 0 \to 1$. For every value of A, we use the model $f(A)$ to find the waveform for this power level, numerically simulate the sampling and quantization, and compute the preprocessor output. To efficiently enumerate all potential outputs, we apply the binary search algorithm described in Algorithm 4-3, and count the resulting number of unique outputs U to determine the effective number of bits as $\log_2(U)$.

Note that the binary search in Algorithm 4-3 assumes a monotonically increasing preprocessor output as A increases. For more complex models $f(A)$ where this does not hold true, the algorithm can be augmented or replaced with a slower linear search that steps through A by sufficiently small ΔA and counts the total number of unique outputs seen.

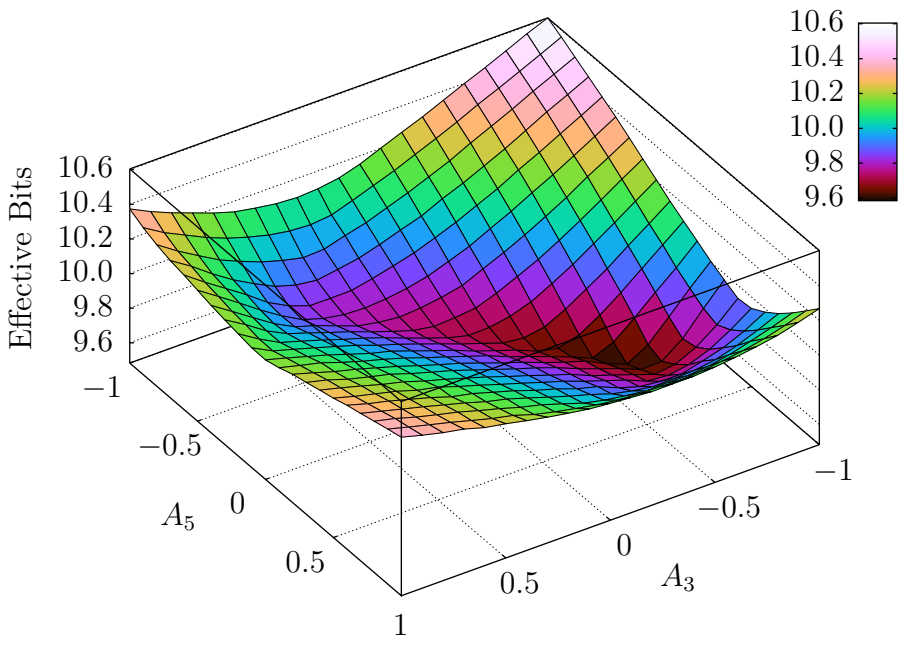


Figure 4-18: Effective bits of preprocessor resolution as relative amounts of harmonic content A_3 and A_5 are varied in (4.49). The harmonics are held at a fixed ratio while the overall waveform scaling is swept from $A = 0 \rightarrow 1$. Sampling parameters are $B = 8$, $N = 128$, and the base case $A_3 = A_5 = 0$ corresponds to 9.8 bits.

The enumeration process was performed for varying combinations of harmonic content ratios A_3 and A_5 in (4.49). Since varying harmonic content can change the peak amplitude of the current waveform, the sampling process was scaled such that the bits-per-amp ratio is fixed for all trials, to maintain consistency. Figure 4-18 shows a plot of the calculated effective bits of resolution. The base resolution is 9.8 bits for $A_3 = A_5 = 0$.

These results demonstrate that adding harmonic content often increases the effective resolution of the preprocessor. Some particular combinations of A_3 and A_5 also reduce it, to a smaller extent. Figure 4-19 shows the waveforms corresponding to the minimum (9.6 bits) and maximum (10.6 bits) resolutions over this parameter range. Informally, the cases where resolution is reduced are those where the total signal amplitude is lowered, or where the waveform has "flat" regions that do not vary much as the parameter A is scaled. Cases where the final waveform exhibits more complexity will generally see improvements in preprocessor output resolution.

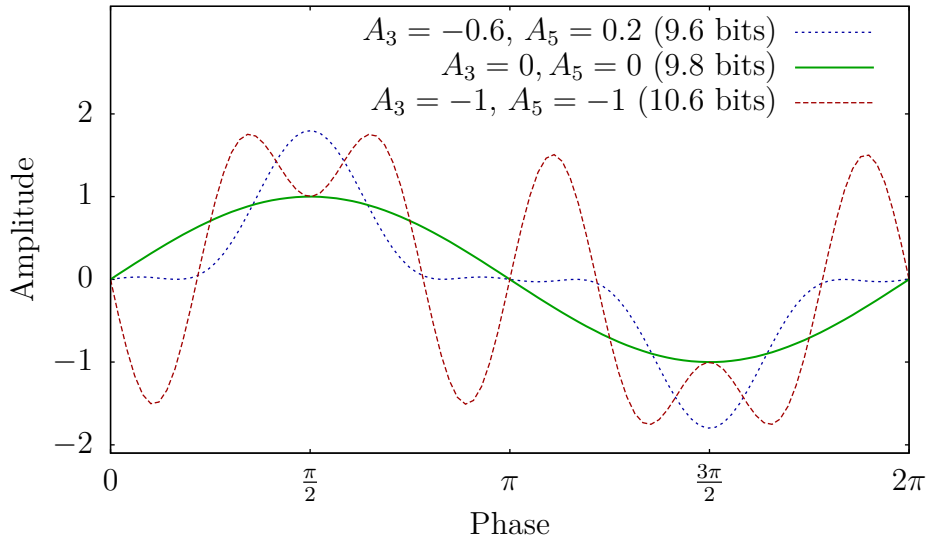


Figure 4-19: Full-amplitude waveforms corresponding to the minimum and maximum resolution from Figure 4-18. In general, increasing complexity increases the effective resolution, although some combinations of harmonics reduce it slightly due to "flat" regions that do not vary much with power scaling.

4.7 Conclusions

The spectral envelope preprocessor accurately extracts relevant harmonic information while providing data storage reduction. The applicability of the preprocessor to complex systems such as multi-phase systems and variable speed drives has been improved by the development of a more flexible and modular "sinefit" preprocessor design with improved phase and frequency estimation. The new preprocessor is designed to integrate with the NilmDB framework and builds upon its ability to correlate, manipulate, and retrieve interrelated data streams.

Chapter 5

Zoom NILM: Physically-Windowed Sensor Architecture

5.1 Introduction

In energy scorekeeping and diagnostic applications, current sensors are often deployed to collect and analyze current waveforms from a collection of loads [71]. Analysis provides load disaggregation and detection, power consumption profiling, and diagnostics based on electrical signatures [7, 9, 56, 71, 72]. Many current sensors are available according to dynamic range and sensitivity. Hall sensors, fluxgate-based sensors, and Rogowski coils have all been used for non-contact current measurement [73–79]. As monitoring systems grow to include more loads and to provide more detailed information about the loads, the scalability and utility of the system depends on the quality of data acquired by the current sensor [13]. When monitoring large loads or collections of loads, some relevant features may be found in harmonic or aperiodic content that is small compared to the current drawn at the fundamental line frequency. For other features, the full amplitude signal may be required.

A physically-windowed sensor architecture is introduced that allows for a more flexible tradeoff between sensitivity and dynamic range. Large-scale variations are cancelled such that the measured signal remains within a small operating window, while the residual small-scale signals are sensed conventionally with an accurate sen-

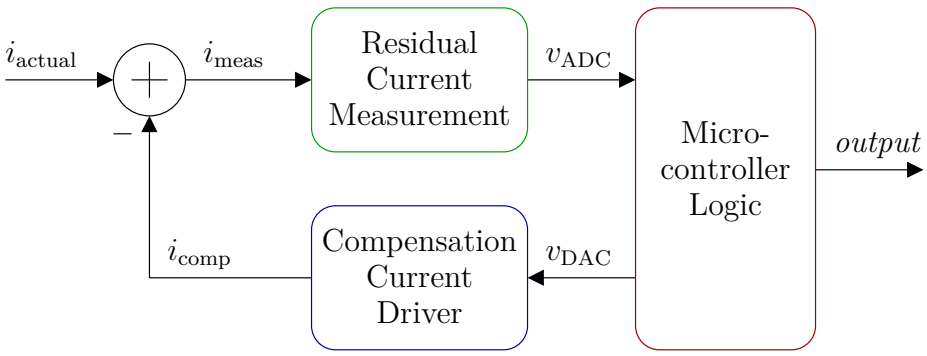


Figure 5-1: System block diagram. The system consists of three primary components: the compensation current driver, the residual current measurement, and the microcontroller logic. The compensation current i_{comp} is subtracted from the primary input current i_{actual} by physical cancellation of magnetic flux.

sor. This architecture is similar to that of pipelined analog-to-digital converters [80], but utilizes a physical cancellation approach that can be applied to magnetic flux-based current sensors, strain gauges, pressure transducers, and many other physical systems. The cancellation is software-controlled by an embedded microcontroller, permitting a variety of windowing techniques and flexible processing and analysis.

This chapter presents an initial application of this concept to power electronics by developing a physically-windowed current sensor that demonstrates high accuracy over a wide input range.

5.2 System Design

Our architecture uses an additional physical input to apply a cancellation signal to a sensor. This enables an accurate but narrow-range sensor to measure effectively beyond its specified operating range. The initial implementation applies this approach to a current sensor and follows the overall design shown in Figure 5-1. A compensation current i_{comp} is driven anti-parallel to large input currents such that the effective total current i_{meas} seen by the sensing element remains within its designated operating range. The microcontroller coordinates and controls the system, performing calibration at startup and adjusting the compensation as necessary to keep the sensor at the desired operating point.

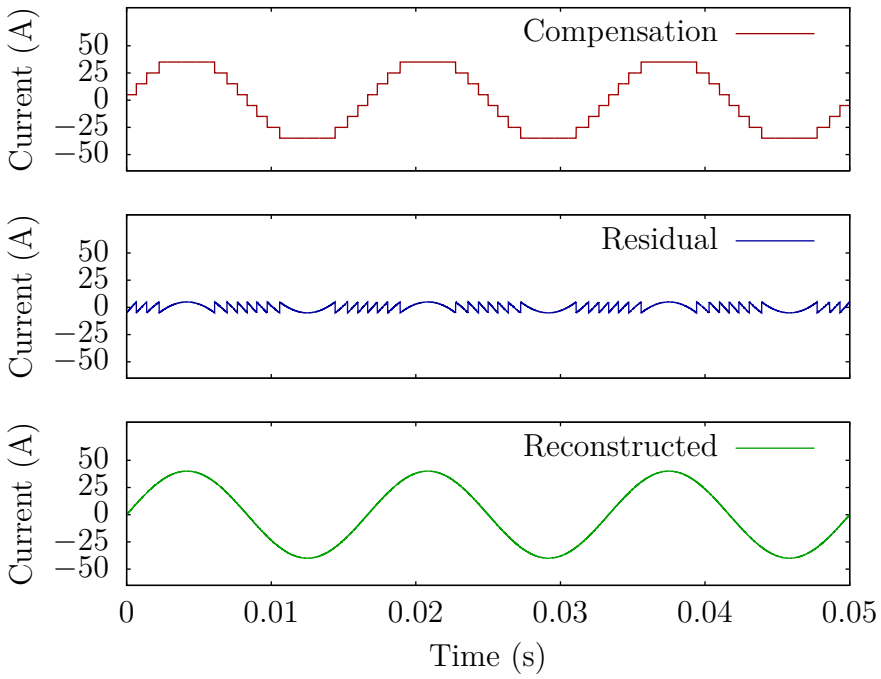


Figure 5-2: Signal reconstruction. The compensation current and measured residual current are combined to determine the total current through the full sensor.

5.2.1 Signal Reconstruction

Figure 5-2 depicts the signal reconstruction used to determine the total current using the windowed measurement. The total input current i_{actual} is calculated from the instantaneous compensation current and sensor measurement as:

$$i_{\text{actual}} = k_c \cdot i_{\text{comp}} + k_m \cdot i_{\text{meas}} \tag{5.1}$$

where k_c and k_m are calibration values determined by physical factors, such as the number of turns on the sensing core and the amount of magnetic coupling.

The compensation current i_{comp} is generated by an operational transconductance amplifier (OTA), detailed in Section 5.4.2. This current i_{comp} is set by the microcontroller using a digital-to-analog converter (DAC) command voltage, v_{DAC}. The total compensation current is given by:

$$i_{\text{comp}} = k_{\text{DAC}} \cdot v_{\text{DAC}} \tag{5.2}$$

where k_{DAC} is determined by the OTA design.

Residual current is measured using a closed-loop Hall sensor, detailed in Section 5.4.1. This current i_{meas} is read from an analog-to-digital converter (ADC) as the voltage v_{ADC}, and is given by:

$$i_{\text{meas}} = k_{\text{ADC}} \cdot v_{\text{ADC}} \tag{5.3}$$

where k_{ADC} is determined by the sensor front-end design. Combining these equations, the complete reconstruction is:

$$i_{\text{actual}} = k_c \cdot k_{\text{DAC}} \cdot v_{\text{DAC}} + k_m \cdot k_{\text{ADC}} \cdot v_{\text{ADC}}$$

The constants can be simplified as:

$$i_{\text{actual}} = k_s(v_{\text{DAC}} + k_r \cdot v_{\text{ADC}}) \tag{5.4}$$

where k_r represents the ratio between the DAC command voltage and the corresponding change in ADC input voltage, and k_s represents a scaling to convert to actual current. This simplified form is used both for discussion and by the internal calibration and windowing procedures described in Section 5.4.3.

5.2.2 Resolution and Range

The performance of the overall physically-windowed sensor system is determined by the parameters of its components. The ranges and resolutions of the compensation current and the residual measurement overlap, as depicted in Figure 5-3. In this example, the ADC is accurate to 11 bits over a range of 5 A, while the DAC command is accurate to 10 bits over a range of 160 A. The amount of overlap directly relates to the parameters k_r and k_s in (5.4).

A key requirement for physically windowed sensing is that the compensation output must remain stable and predictable to the full system resolution. In Figure 5-3,

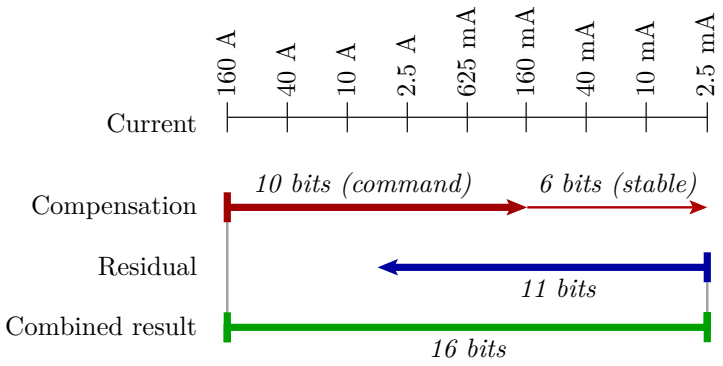


Figure 5-3: Overlap of the DAC output, relating to compensation current, and the ADC input, relating to the residual measurement. The ranges and overlapped positions are related to the parameters in (5.4). The combined result shows high accuracy over the full range.

this requirement is depicted as a dashed line on the DAC output. Here, for the lowest-order bits of the combined result to be accurate, each of the 2^{10} possible DAC commands must result in a voltage stability of one part in 2^{16}. Certainly, a 16-bit DAC would suffice. However, only stability is needed, not accuracy. If a lower-resolution DAC is, or can be made to be, similarly stable in output, it is sufficient for the sensor architecture. Using such a DAC may provide cost or performance benefits.[1]

Given that the stability requirement is met, then the actual output voltage v_{DAC} can be related to the DAC command x as:

$$v_{\mathrm{DAC}}(x) \propto \frac{x}{2^{10}} + \frac{\textsc{lookup}[x]}{2^{16}} \tag{5.5}$$

where $\textsc{lookup}[x]$ is a 2^{10}-entry table that stores these 6 extra stable bits. This table can be populated by the microcontroller in a calibration step that uses the ADC input to determine the low-order bits of each DAC output.

[1]The prototype implementation in Section 5.4 simulates this stability by using a 16-bit DAC with fixed random low-order bits on a 10-bit command. The low-order bits are set by the microcontroller and can be adjusted for testing purposes. The system has also been demonstrated with a true 10-bit DAC.

5.2.3 Windowing

The front-end current measurement is "windowed" by the compensation current in the sense that the compensation sets a particular operating point, and the Hall sensor measures a small window of current around this point. The microcontroller has significant flexibility in the windowing approach, and the behavior can be adjusted based on expected workloads and system parameters.

A basic approach to windowing is to continuously recenter the window so that the ADC measurement is zeroed; that is, the residual current is driven to zero after each sample. However, this requires that the OTA change its current output nearly continuously as the input signal changes, increasing the bandwidth requirements and potentially making the data less accurate if changes in compensation current are slow to settle.

The approach demonstrated by the reconstruction in Figure 5-2 is to change the DAC command when the residual current in the sensor approaches the limits of the front-end. The compensation current will remain constant for small input signal changes, and only change for larger input signals that exceed the window. For many input signals, this may allow the compensation to change relatively slowly, reducing bandwidth requirements for the compensation driver.

More advanced approaches are possible, particularly for loads with known characteristics. A predictive estimator in the microcontroller can perform an anticipatory change in the compensation current so that the residual sensor current would be expected to fall within the sensor limits at the next sample interval. Such techniques can potentially increase the slew rate capability of the system.

5.2.4 Bandwidth

The bandwidth of the physically windowed sensor system depends on the input signals and their relation to the sensor window. There are two fundamental regions of operation: the first, within the windowed range of the residual current measurement, and the second, over the full range of the compensation current. For input currents

that fall entirely within the window, the bandwidth performance of the system is equal to that of the residual current sensor front-end, as the compensation current is held constant. For full-scale input signals, the bandwidth is instead limited by how fast the compensation current can track the input change.

Maximum slew rate may be further affected by the windowing algorithm in use. Once the residual current exceeds the range of the sensor window, the compensation command must be adjusted. In the absence of prediction, the microcontroller will not know by how much the residual current exceeded the window, and will be limited to stepping the compensation by one "window" worth of current at a time. This, combined with the sampling rate of the residual sensor and the bandwidth of the compensation driver, will set the maximum $\frac{di}{dt}$ that can be accurately tracked. For slew rates outside this limit, the subsequent front-end sample will still exceed the window, and the microcontroller can report the potential inaccuracy as part of the output data stream.

The bandwidth and slew rate limits are a function of the resolution, range, and bandwidth of the system components. Flexible tradeoffs can be made by, for example, adjusting the system to increase k_r in (5.4). This would have the effect of increasing the relative size of the sensor window, increasing the region in which the recorded signal retains full bandwidth, and increasing the maximum slew rate. Conversely, increasing k_r increases the overall resolution of the reconstructed signal.

5.3 Benefits and Motivation

In many physical systems, large-scale changes occur at relatively slow speed while small-scale details can change rapidly. For example, an electric motor draws a 60 Hz fundamental current from the utility, but it may be desirable to observe a principal slot harmonic (PSH) at several hundreds or thousands of Hertz to track the motor speed [81]. These small, high-frequency details are superimposed on top of the 60 Hz current and need to be examined without saturating the sensor front-end. Conventional current sensors like the closed-loop Hall-effect sensor utilize a single compen-

sation circuit to measure current. The physically-windowed sensing system, instead, divides the measurement into two subsystems, the compensation current driver and the residual current measurement. By dividing the problem and taking advantage of the fundamental differences between the requirements of the large-scale and small-scale measurements, the windowed system can utilize power and bandwidth trade-offs in the design of design each subsystem. This section describes these trade-offs and their design considerations.

5.3.1 Resolution

To obtain an accurate measurement, the system needs a reference that is stable to the required resolution specification. A conventional current sensor can utilize a single high-resolution ADC to perform the measurement, or it can use a single high-resolution DAC as a reference against which to compare a measurement. In both cases, it is required that the ADC or DAC be both stable and accurate to the full resolution.

With the physically-windowed approach, it is sufficient that the DAC be stable, but not necessarily accurate. A DAC with fewer controllable bits, but stable to the full resolution, can still be used. Initial experiments, testing the output voltage of a 16-bit AD7846 DAC with a HP34401A multimeter, demonstrated an accuracy of approximately 26 μV on a ±5 V range, or approximately 18.5 bits, in a controlled environment. This example shows that, under some conditions, the output of the DAC is more stable than the controllable input.

The physically-windowed sensor design can then use a moderately accurate DAC and a moderately accurate ADC to create a compound data acquisition system that can accurately resolve more than the number of bits provided by either the DAC or ADC alone. For example, assuming proper calibration is performed, the system may be able to use one 10-bit DAC and one 10-bit ADC to create an effective 12-bit data acquisition system.

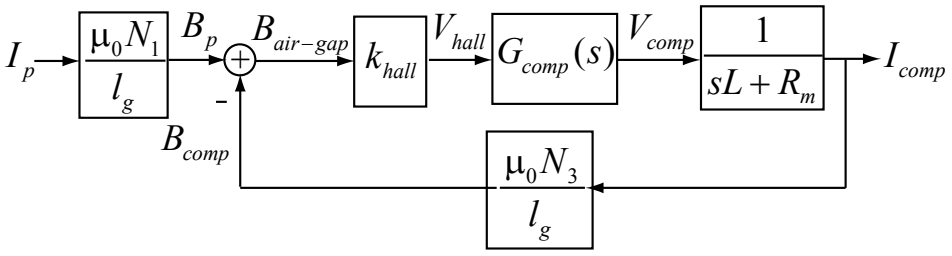


Figure 5-4: Typical design for a closed-loop hall sensor, with one compensation circuit.

5.3.2 Bandwidth

A key element in the design process for analog circuits is the trade-off between power consumption and bandwidth. If the signal of interest is comprised of both low-frequency and high-frequency content, the current sensor may be able to take advantage of this separation by utilizing two separate compensation circuits, each optimized for one frequency region.

The basic topology of a typical current sensor based on zero-flux sensing consists of three parts: a magnetic flux sensor, a compensation circuit, and a compensation winding. The system block diagram is shown in Figure 5-4. The input current creates a magnetic flux which is focused in the air gap of the gapped magnetic core. The magnetic flux sensor senses any magnetic flux in the air gap, and provides an output signal for the compensation circuit. This circuit drives a cancellation current to cancel the magnetic flux. Effectively, the system forces the magnetic flux in the air gap to zero, keeping the magnetic core around the zero-flux operating point and away from the saturation region.

In order to measure a fast dynamic signal accurately, the measuring system is required to have a wide bandwidth and a large open-loop gain. In another words, the gain-bandwidth product of the open-loop transfer function must be very large. As shown in Figure 5-4, the system can be designed to meet the large gain-bandwidth product by choosing an appropriate compensation circuit $G_{\text{comp}}(s)$. A typical compensator is an integrator with a lead compensation. The integrator provides a large open-loop gain at low frequency, whereas the lead compensation provides stability

for the system. A typical transfer function of the lead-compensated integrator circuit can be described as:

$$G_{comp}(s) = \frac{(s\tau_2 + 1)}{(s\tau_1 + 1)} \tag{5.6}$$

where $\frac{1}{\tau_1}$ is a low frequency pole of a practical integrator circuit, and $\frac{1}{\tau_2}$ is the compensated zero. In this case, we assume that all high frequency poles of the op-amp and parasitics are negligible at the cross-over frequency. The compensation circuit would be designed to meet the required bandwidth of the input signal.

The compensation circuit must cancel the flux by driving an equivalent current through the magnetic core, typically using an output stage consisting of an operational amplifier and push-pull buffer circuit. If the compensation current coil consists of N_3 turns, the primary winding consists of N_1 turns, and the maximum input current is $i_{\text{actual\_max}}$, then the compensation circuit must be able to drive $\frac{N_1}{N_3} i_{\text{actual\_max}}$ through the coil.

As N_3 is increased, the drive current is lowered, but this also affects the inductance that the compensation circuit has to drive. Specifically, the inductance of the compensation coil is proportional to ${N_3}^2$. Large inductance will limit the maximum slew rate that the buffer circuit can provide for compensation. The slew rate $\frac{di}{dt}$ of the current in an inductor is given by:

$$\frac{di}{dt} = \frac{v_L}{L} \propto \frac{v_{supply}}{{N_3}^2} \tag{5.7}$$

where v_L represents the voltage across the inductor, which is limited by the supply voltage of the system. Given the same supply voltage, a smaller inductance will allow the system to follow the input current more accurately.

The physically-window sensing system, on the other hand, divides the compensation circuit into two parts: a large, slow compensation circuit for the bulk of the flux cancellation, and a small, fast compensation circuit for measuring the residual. Block diagrams of the proposed system are shown in Figure 5-5.

For the smaller residual measuring range, the compensation circuit can have the

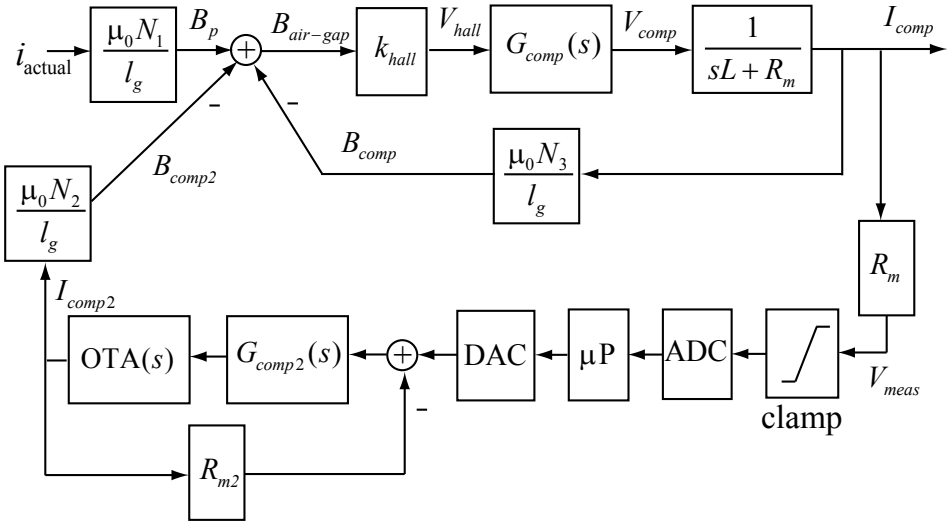


Figure 5-5: Physically windowed current sensor, with two compensation circuits.

same design as the single compensation circuit case. Since the maximum current output is reduced, the number of winding turns N_3 can be reduced, effectively lowering the inductance of the coil. The lower inductance allows this compensation circuit to follow the higher $\frac{di}{dt}$ rate according to (5.7), given the same supply voltage.

The auxiliary compensation circuit must provide a cancellation current for a larger portion of the magnetic flux in the core. Specifically, the auxiliary compensation circuit is responsible for providing a cancellation current of $\frac{N_1}{N_2} i_{\text{actual\_max}}$ where N_2 is the number of turns in the auxiliary winding. This is analogous to the requirement of the first compensation circuit, and the circuit can be made similarly, with minor adjustments. In our prototype, the auxiliary compensation circuit is implemented with a current source, which provides a high output impedance as seen by the other loop across the current transformer. As a result, when the auxiliary compensation circuit is providing a constant current output, interaction between the two compensation circuits is minimized.

By separating the compensation current into two subsystems, the design process can be divided into two problems which may be tailored to take advantage of the input signal characteristics. In this case, two compensation circuits add complexity

to the system, but provide more flexible power and bandwidth trade-offs.

5.3.3 Feedback

Feedback is one of the key concepts that enables the proposed sensing system to work properly. This proposed system consists of two analog feedback loops and one digital feedback loop. The small compensation current uses analog feedback to keep the magnetic core in the zero-flux region. The second feedback loop ensures an accurate conversion between the command voltage and the compensation current. Both of these feedback loops act as minor loops within the digital feedback loop. The digital feedback enables flexible control of the entire process, and can be easily adapted for different input signal characteristics.

5.4 Prototype Implementation

The prototype system was implemented according to the design introduced in Section 5.2. The system block diagram is shown in Figure 5-6. The physical coupling of the subsystems occurs on a single toroidal core. The primary current to be measured passes through N_1 turns on the core. The cancellation current passes through N_2 turns, wound in the opposite direction. The residual current is measured by a closed-loop Hall sensor that utilizes N_3 additional turns. The N_2 and N_3 loops are co-wound to minimize leakage inductance. Typical values for our testing are $N_1 = 50$ and $N_2 = N_3 = 200$.

5.4.1 Residual Current Measurement

The residual current measurement is based on the closed-loop Hall-effect sensor shown in Figure 5-4. The sensor drives an output current such that the flux perceived by the Hall element is near zero. At this operating point, the temperature drift and offset of the Hall sensor are minimized. The output current is read by an 11-bit ADC to produce the residual measurement.

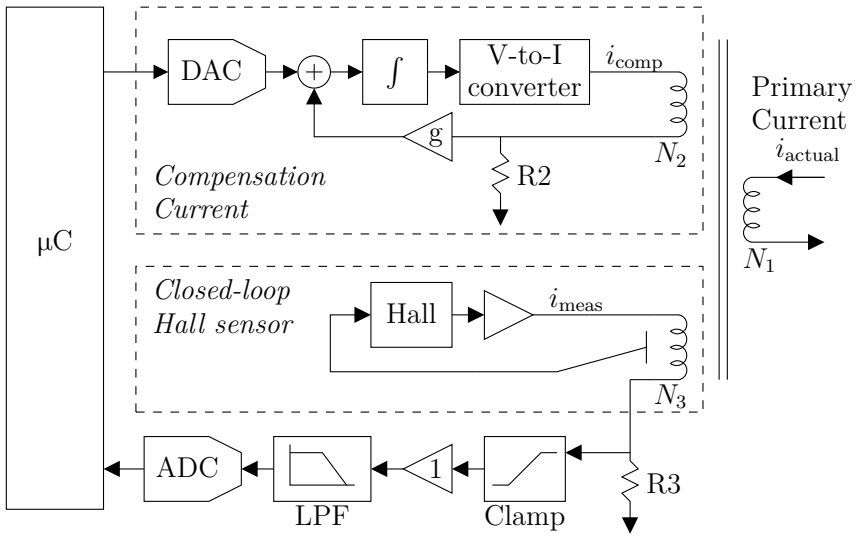


Figure 5-6: Detailed block diagram of the prototype microcontroller-based system.

In our system, the Hall sensor is designed to measure over a small current range of approximately ±2.5 A. If the input signal starts to exceed this range, the compensation current driver is separately commanded to cancel a portion of the flux in order to keep the residual sensor operating normally.

Figure 5-7 shows measured calibration curves between the compensation current output and residual sensor input, for various operating points set by the primary current. The slope of each line corresponds to the constant k_r in (5.4). The dashed lines indicate the approximate measuring range of the ADC for the residual current. The digital controller attempts to maintain the window so that the residual current always falls within this range.

During a high current transient, the input current may temporarily exceed the ability of the system to compensate. To prevent the residual sensor from overloading in this condition, a clamp circuit is added at the output of the Hall sensor. This extra clamp current serves to cancel the primary current and limit maximum residual. The clamp current is not reflected in the ADC reading, causing the measured values to saturate, as shown shown in Figure 5-7. After the overload condition, the clamp deactivates and the Hall sensor returns to normal operation. The system utilizes a magnetic core with a low remnant flux to further minimize the offset error after experiencing such a transient.

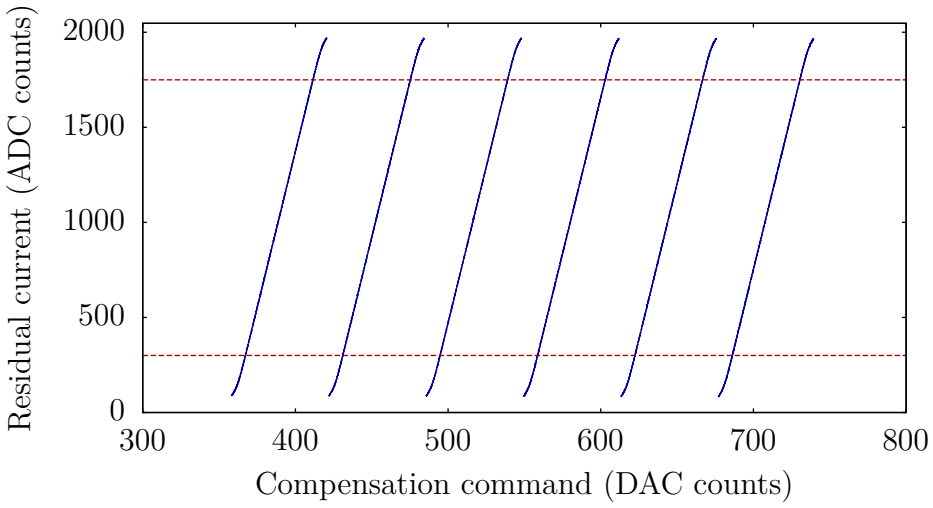


Figure 5-7: Calibration curves showing 11-bit residual current measurement versus 10-bit compensation current command, at various fixed primary currents. The curves are generally linear with slope k_r and flatten out as the clamps begin to activate outside of the dashed horizontal lines.

5.4.2 Compensation Current

The compensation current driver uses a highly stable digital-to-analog converter to establish a voltage command reference. A closed-loop circuit, shown in Figure 5-6, is designed to scale and convert the voltage command into the desired output current. This output current is co-wound on the core with the output from the closed-loop Hall sensor circuit. To minimize interaction between the two feedback loops, the output stage of the compensation circuit is high-impedance and appears as an open circuit to the residual sensor circuit.

The OTA design is shown in Figure 5-8. In this implementation, the OTA includes a voltage buffer front-end to receive the voltage command from the compensation current feedback op-amp. The buffered command is used to establish a reference output current in the second stage. This reference current is replicated through a current mirror structure. The current mirror uses a cascode topology to improve the output impedance. The emitter degeneration resistors are added to scale the current and to prevent a thermal runaway condition. The OTA structure includes multiple output branches connected in parallel, in order to minimize the power dissipation

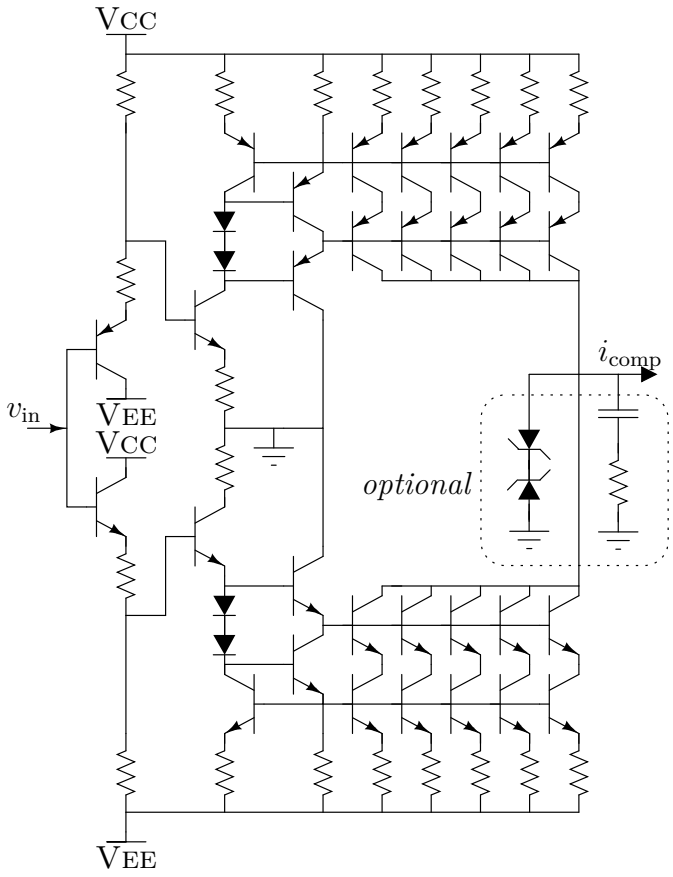


Figure 5-8: Output stage of the compensation current subsystem, showing the OTA implementation. A complete schematic, including component values, can be found in Appendix G.3.

per branch. Within each cascode branch, the transistor next to the rail sets up the mirrored current, and the cascode transistor acts as a current buffer. The power dissipation in each cascode branch will be concentrated at the cascode transistor. Therefore, the thermal effects on the mirrored current are reduced. Finally, the β-helper transistors are included to provide additional base current for the output stage.

The OTA output passes through N_2 turns on the core and is measured with a sense resistor. The analog feedback loop in the compensation current subsystem serves to minimize error within the output stage. The output current measurement can optionally be provided back to the microcontroller through a low-bandwidth 24-bit ADC for calibration purposes.

5.4.3 Microcontroller

Control logic for the prototype is implemented using a Microchip dsPIC33FJ256GP-710 microcontroller. It controls the sampling of the residual current measurement, implements the windowing algorithm used to set the compensation current, and communicates all data to a computer for analysis. The microcontroller also performs calibration at startup and on request.

5.4.3.1 Sampling

The ADC is sampled at 8 kHz, a rate chosen to match that used in existing non-intrusive load monitoring systems [13]. At each sample, the microcontroller calculates the total reconstructed current from the compensation command and the residual measurement, and transmits this data to the computer. If necessary, the compensation command is then changed to adjust measurement window.

The sampling interval has a direct influence on the slew rate capability of the system. If the residual current exceeds the window range, the measurement is clamped and the recombined output will be inaccurate. For a given operating point and windowing strategy, a maximum current excursion i_w can be observed within the current window before saturation. Given the sampling interval Δt and primary input slew rate $\frac{di}{dt}$, it is necessary that:

$$\Delta t \cdot \frac{di}{dt} < i_w \tag{5.8}$$

Thus, decreasing Δt, by increasing the sampling rate, has a corresponding linear effect on the maximum slew rate $\frac{di}{dt}$.

In practice, the limit on Δt is set by the response time of the hardware components and the processing speed of the microcontroller. Regardless of the sampling rate, the data reporting rate to the computer can be maintained at 8 kHz for compatibility.

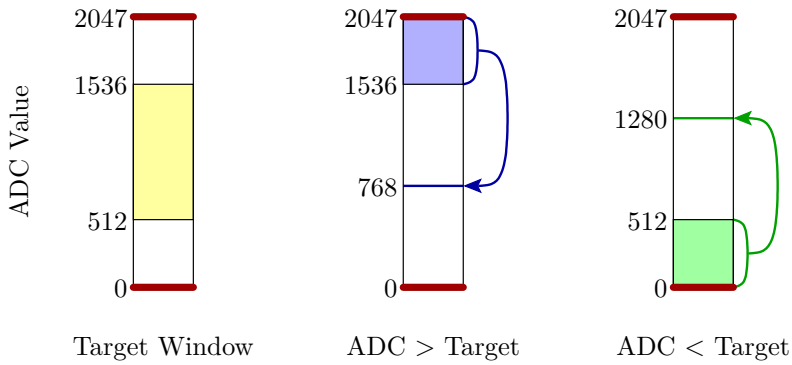


Figure 5-9: Windowing approach in the prototype implementation. For input ADC values greater than 1536, the compensation current is changed such that the target ADC value is 768. For input ADC values less than 512, the target ADC value is 1280. Hardware clamps are active near the limits, 0 and 2047.

5.4.3.2 Windowing Strategy

Using (5.4), the microcontroller can determine by how much a given change in DAC command will affect the sensor measurement at the ADC input. Assuming that the primary current i_{actual} remains constant, a pair of DAC and ADC values are related by:

$$k_s(v_{\text{DAC},1} + k_r \cdot v_{\text{ADC},1}) = k_s(v_{\text{DAC},2} + k_r \cdot v_{\text{ADC},2}) \tag{5.9}$$

$$v_{\text{DAC},1} - v_{\text{DAC},2} = k_r(v_{\text{ADC},1} - v_{\text{ADC},2}) \tag{5.10}$$

Thus, to cause a change of ΔADC at the residual measurement, the DAC command should be changed by $k_r\Delta$ADC. The prototype implementation uses this approach to "recenter" the window whenever the ADC value begins to approach the clamp limits. This is depicted in Figure 5-9. For example, when a sample x from the ADC exceeds a fixed upper limit of 1536, the compensation command is increased by $k_r(x - 768)$ so that the next sample is near 768.

The chosen target ADC values intentionally overshoot the center of the ADC range because it is expected that, in most cases, an increasing current will continue to increase. This provides some extra "headroom" for the common case, which in turn increases the maximum slew rate that the prototype can handle.

5.4.3.3 Communication

All data, including raw DAC and ADC values and calibration constants, are continuously sent to a computer via a full-speed USB link. In some configurations, the microcontroller may read the ADC more frequently than the samples are sent to the computer, and so status flags that indicate error states are also included independently. For example, one flag denotes whether the ADC value was ever observed in the clamped region, which indicates that the returned data for that sample may not be accurate to full resolution.

5.4.3.4 Calibration

In order to perform windowing accurately, the microcontroller needs to know the calibration constant k_r (from Section 5.2.1) and the table LOOKUP (from Section 5.2.2).

The value of k_r can be determined using the relationship in (5.10), if the primary current is constant. In the prototype implementation, the microcontroller assumes constant current and performs calibration at startup, or when triggered by the connected computer. Using an initial estimate $k_r = 1$, the calibration algorithm adaptively adjusts the estimate as it changes v_{DAC} to seek two specific v_{ADC} values corresponding to ADC inputs 512 and 1536. Once these DAC commands are found, each v_{ADC} is oversampled to reduce noise and a final accurate estimate of k_r is computed.

Table LOOKUP is used to map each low-resolution DAC command to its corresponding high-resolution stable output voltage. In the current prototype, this extended stability is simulated using an accurate DAC, and so the lookup is hard-coded to match the randomized low-order bits written to the DAC. However, the system supports an additional low-bandwidth 24-bit ADC for measuring the output compensation current. This calibration ADC could be used to measure and fill the LOOKUP entries for each possible DAC output, in the absence of a hard-coded table.

Finally, to convert the final output of the physically-windowed current sensor to amperes, the scaling factor k_s is used. It is not directly needed by the microcontroller

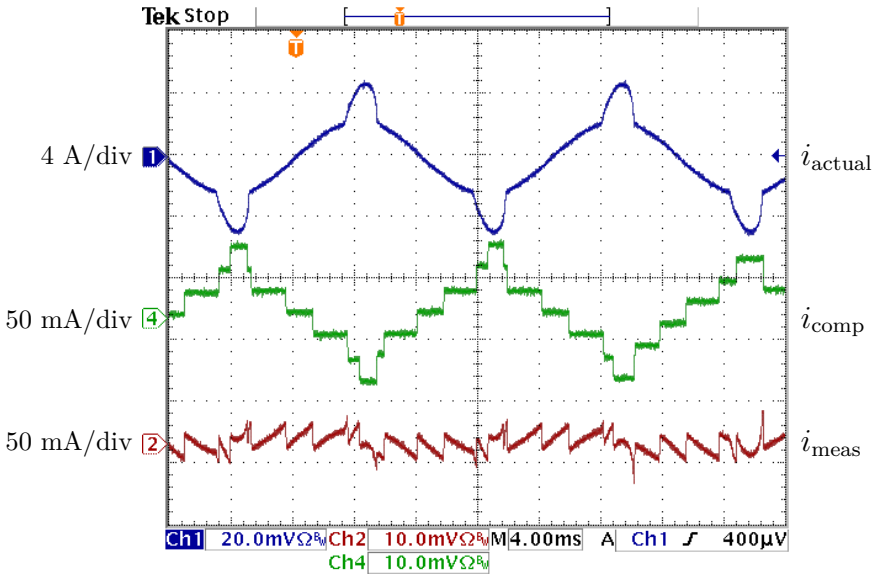


Figure 5-10: Oscilloscope traces showing the measured primary, compensation, and residual currents while the prototype physically-windowed current sensor is in normal operation.

logic, and is currently calibrated by the computer in post-processing using known test values from a Keithley 2400 Sourcemeter.

5.5 Prototype Results

Various aspects of the prototype physically-windowed current sensor system have been tested.

5.5.1 Full System Functionality

Basic functionality was tested by constructing a test load consisting of an incandescent light bulb and a personal computer, which together draw power at both the fundamental and third harmonics of the line frequency. Figure 5-10 shows the waveforms as measured by external test equipment. The input current is i_{actual}, the generated compensation current is i_{comp}, and the residual current is i_{meas}.

The reconstructed output of the sensor system for the same test load, based on data reported by the microcontroller, is shown in Figure 5-11. In some cases, the high slew rate associated with the third harmonic content in the load caused the residual

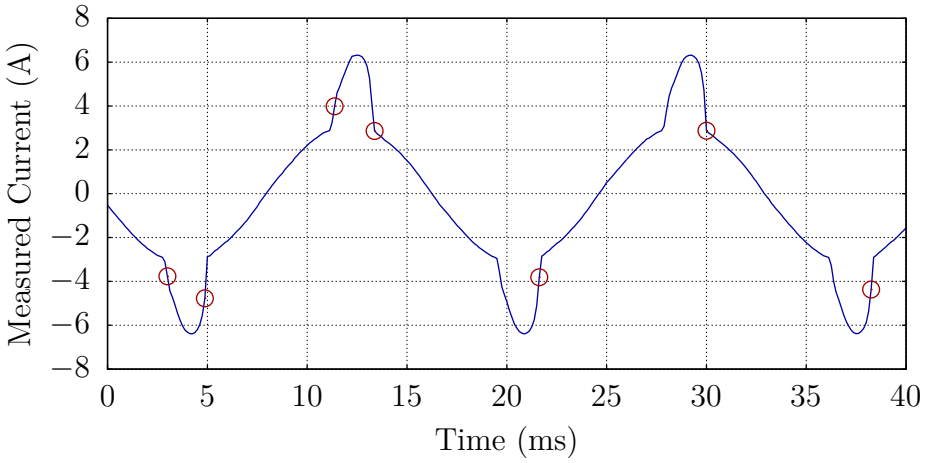


Figure 5-11: Reconstructed current from the sensor. Circled points indicate individual samples that are known to be potentially inaccurate because the residual current was in the clamped region of the sensor window.

current measurement to exceed its window and enter the region where the clamps are active. The recombined data at these samples is known to be potentially inaccurate because of this. In the figure, these specific samples are circled. Note that they do not occur uniformly on every line cycle, because such excursions from the window depend on the varying relationship between sample time, slew rate, and the current window position.

5.5.2 Static Resolution Tests

The maximum resolution of the sensor system was characterized by analyzing the static performance. For this test, a Keithley 2400 Sourcemeter was used to supply various dc currents. These currents were passed through the sensor $N_1 = 50$ times to create the primary current i_{actual}. Each effective current level was chosen at random from a ± 25 A range. Once the test current stabilized, the reconstructed output from the prototype system was sampled at 8 kHz for approximately 110 ms. Approximately 500 unique test current levels were applied in total. A histogram of the resulting error between the Keithley reported output and the reconstructed sensor output is shown in Figure 5-12. Typical errors for each sample are less than ± 5 mA. Over the full 160 A range of the compensation current driver, this 10 mA range translates into an

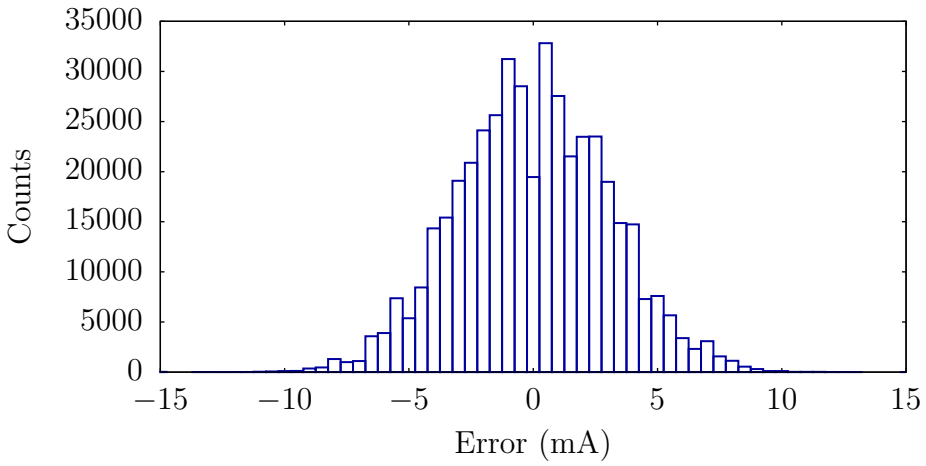


Figure 5-12: Histogram of the measured errors during a test of dc performance. 91% of the samples fall within ±5 mA, which gives an accuracy over the full ±80 A range of approximately one part in 2^{14}.

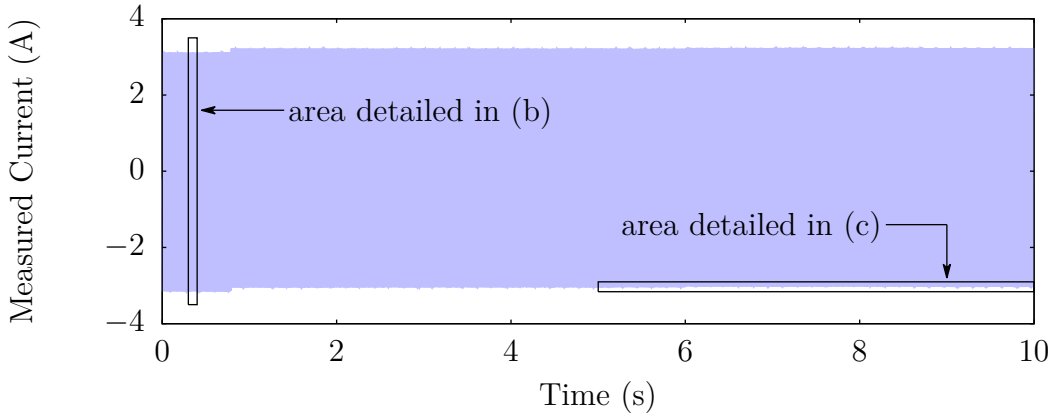
effective resolution of

$$\log_2(160/0.01) = 13.996 \text{ bits} \tag{5.11}$$

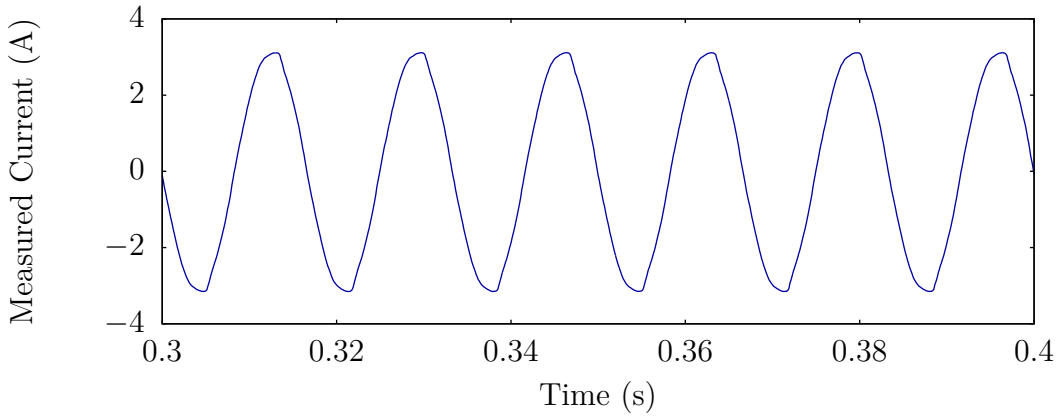
In the prototype system, the DAC command for the current compensation is 10 bits, while the ADC input from the residual measurement is 11 bits. This result of nearly 14 bits demonstrates the concept of using physical windowing to extend the sensor resolution over a larger range.

5.5.3 Dynamic Resolution Tests

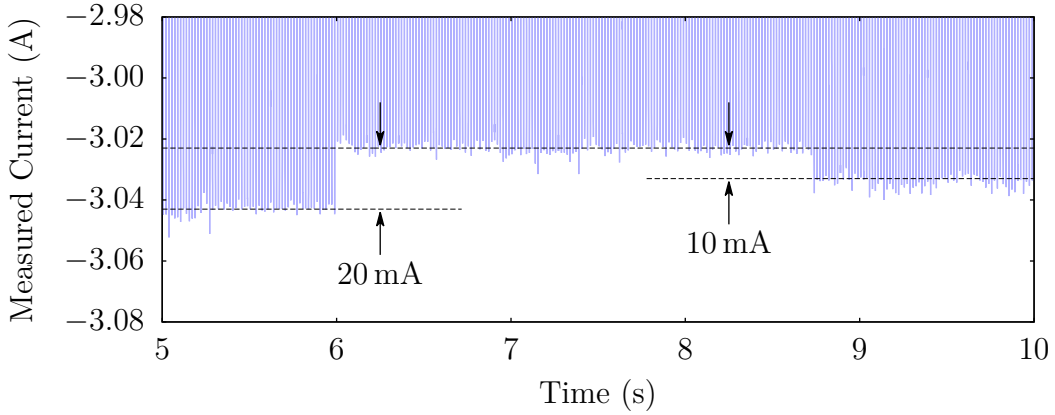
The ability to resolve small signals while tracking a large signal was evaluated by creating a test load consisting of an incandescent lamp bulb in parallel with the Keithley 2400 Sourcemeter. For this test, the change in the envelope of the measured lamp ac waveform was examined as various small test currents were injected through the sensor using the Keithley source. The injected current was cycled between 0 mA, 20 mA, and 10 mA. The resulting reconstructed waveform is shown in Figure 5-13. The small change in dc level can be seen in the detail shown in part (c). Like the static resolution test, this shows a resolution of approximately 10 mA.



(a) Full reconstructed current.



(b) Detail over several line cycles.



(c) Detail of a small dc change.

Figure 5-13: Example of measuring a small dc current on top of a large ac current. The ac current is from a light bulb and the dc offset is added by a Keithley 2400 Sourcemeter. The two currents are passed in parallel through the sensor core. Steps of 10 mA are resolvable in the reconstructed sensor output.

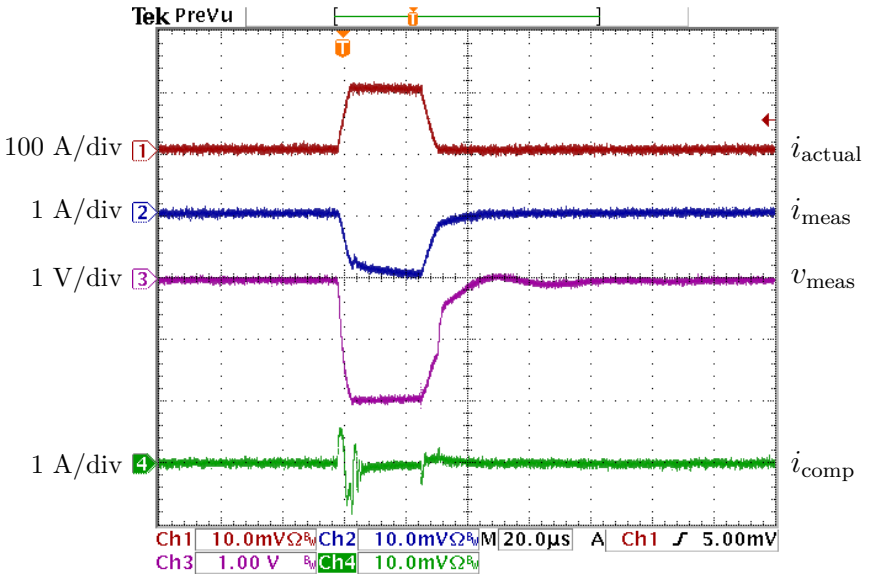


Figure 5-14: System response during a high current transient. A pulse of primary current i_{actual} =100 A is applied for 25 µs. The clamp activates, limiting the residual current in the core. After deactivating, the offset measured at the core is less than 5 mV. The compensation current is also shown, demonstrating that it quickly stabilizes back to the commanded value after an induced change.

5.5.4 Residual Sensor Transient Response

In order to prevent drift in the output, the residual sensor employs a current clamp that prevents the toroidal core from saturating. The transient response of the residual sensor was tested by applying a large instantaneous current of 20 A through $N_1 = 5$ turns. This results in an effective primary current of 100 A, significantly greater than the typical operating window for the residual. This test simulates the case where a large input transient is seen before the compensation current is applied. The results are shown in Figure 5-14. During the transient, the clamp activates, providing necessary current to keep the core near the zero-flux operating point. When the transient disappears, due to either a change in input or a change of the compensation current, the clamp deactivates and the residual current measurement resumes normal behavior. The offset drift in the residual sensor output V_{meas} is less than 5 mV after the transient, demonstrating that hysteresis loss in the core was minimized.

5.6 Conclusions and Further Work

The prototype sensor has demonstrated that the physical windowing approach is able to extend the range of an 11-bit current sensor with a 10-bit compensation command to provide approximately 14-bit resolution in the reconstructed output.

This windowing technique has many uses in smart grid and other applications where there is a need to measure fast, small signals on top of slow, large signals. The non-intrusive load monitor suggests many example uses, such as finding the high-frequency principal slot harmonics for motor diagnostics, or examining very small loads in aggregate power metering. The same technique is expected to be applicable to other physical systems, such as pressure monitors and strain gauges, with example uses including water system monitoring and wing flutter measurement.

Chapter 6

Conclusions

The non-intrusive load monitor has widespread applications and benefits. It provides the ability to accurately identify and track the behavior and state of multiple electronic and electromechanical systems without requiring physical modification or instrumentation of each load. This work has presented a new, comprehensive framework for improving the quality and capabilities of the NILM, while introducing a powerful new system architecture to manage and analyze the vast quantities of data it generates.

6.1 NilmDB Framework

The NilmDB data storage and management framework represents a transformative shift in the design and implementation of load monitoring systems. It provides a fully structured, consistent, network-aware architecture that flexibly enables the development of actionable diagnostics across a wide variety of systems. NilmDB organizes and standardizes the collection and processing steps, enabling modular and reusable filter components to streamline and simplify the deployment of monitoring systems. A wide selection of fully tested, documented components is provided as a solid baseline for NILM systems.

With NILM Manager, non-intrusive load monitoring gains an unparalleled level of accessibility. A user anywhere in the world can instantly and easily visualize

and navigate NILM data, from the original raw waveforms to high-level reports and health indicators, with nothing more than a web browser and Internet connection. The configuration, status monitoring, and process management provide user-friendly control, while simultaneously exposing all of the power and capabilities of NilmDB through the filter interface.

Together, NilmDB and the NILM Manager provide the solution to the "big data" analytics problem of large-scale power system monitoring. They enable modern advanced NILM techniques through high-quality, high-speed data acquisition, while maintaining low network bandwidth requirements and flexible computing options. Remotely reconfigurable workflows enable an "install-once" approach where new diagnostic algorithms can be deployed and developed on any live system at any time, delivering actionable results exactly when and where they are needed.

6.2 Preprocessor

The spectral envelope preprocessor has long been one of the pillars of non-intrusive load monitoring of ac loads. The new "sinefit" spectral envelope preprocessor developed in this work greatly improves the implementation of this fundamental operation. Variable and noisy frequencies are tracked more accurately, and the system recovers fully from temporary service interruptions. Through integration with the NilmDB framework, the preprocessor offers accurate timestamping of all data and a modular design that is easily applied to extracting both and voltage current harmonics. Configurable phase correction extends the preprocessor to multi-phase systems and increases accuracy with current transducers and other acquisition hardware that introduces phase shifts.

One of the most useful features of spectral envelopes is their ability to extract and represent the useful information contained in low-resolution and noisy signals. The effective resolution of the preprocessor is directly related to the applicability of load monitoring for systems of aggregate loads. Here, the resolution of the sinefit spectral envelope preprocessor output has been fully explored, including for waveforms with

significant harmonic content and added noise. These results give clear guidelines on how data acquisition and load scaling can affect the accuracy of monitoring and diagnostics.

6.3 Physically Windowed Sensors

The physically windowed sensor architecture introduced a system that provides a significant improvement in sensor resolution across a wide variety of physical systems. This windowing technique is particularly suitable for NILM applications, where small-scale and large-scale signals are aggregated together and either overwhelm or are missed by conventional data acquisition systems. An implementation of the system for contactless current measurement was developed, demonstrating the high combined performance and data quality that can be attained with cost-effective components. The system architecture matches up well with the multi-stream capabilities of NilmDB and will help extend NILM techniques to ever-larger and more complex systems.

6.4 Future Work

Future NILM research and development is well-suited to be performed entirely within the context of the NilmDB framework. The tools and software provided in this thesis create a complete NILM system that is ready for deployment and use. The functionality of NILM Manager is rapidly increasing and will allow new data analytics to be easily designed, tested, and deployed. At the same time, NilmDB is being applied to new smaller, cheaper acquisition and processing hardware. It is hoped and expected that these tools will significantly lower the barrier to entry for the development of an increasingly capable and powerful NILM.

Appendix A

NilmDBuntu System Image

A.1 Overview

NilmDBuntu is a complete operating system and software distribution based on the Xubuntu version of Ubuntu. It is designed to provide an easy and reproducible way to build a NilmDB monitoring system from scratch. It contains the following pre-installed and pre-configured components:

- NilmDB 1.9.7
- NilmRun 1.3.3
- NilmTools 1.4.10
- Ethstream 1.3

These components are configured and installed as described in Section 3.1.4. NilmDBuntu also includes a wide variety of tools useful for further NILM filter and software development, such as NumPy, SciPy, GNU Octave, Gnuplot, IPython, and Matplotlib.

The NilmDBuntu installation images are ISO images that can be written directly to a DVD or USB key. There are two available versions, which are largely identical, but based on different underlying versions of Xubuntu. This may be useful for situations where the target system hardware is better supported by one of the two

versions. NilmDBuntu 12.10.1 is recommended, as its underlying Xubuntu release is supported for a longer period of time. The following images are available:

NilmDBuntu 12.10.1, based on Xubuntu 12.10:

```
http://bucket.jim.sh/nilmdb/nilmdbuntu-12.10.1.iso
```

NilmDBuntu 13.04.1, based on Xubuntu 13.04:

```
http://bucket.jim.sh/nilmdb/nilmdbuntu-13.04.1.iso
```

A.2 Installation

Installation requires a 64-bit (`amd64`) system. The NilmDBuntu image should be written to a DVD or USB key, and the target system configured to boot from this media. At boot, a menu similar to that shown in Figure A-1 will be shown. Select "Boot NilmDBuntu" to load the system.

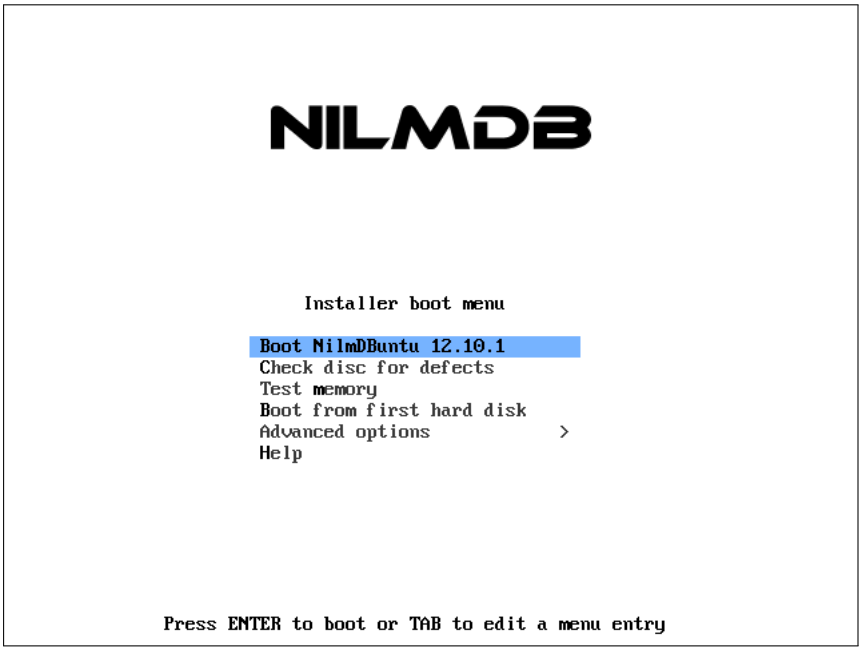


Figure A-1: NilmDBuntu installation boot menu.

NilmDBuntu will boot into a live system that mirrors how the installed system will appear, as shown in Figure A-2. From here, the various NilmDBuntu software components can be tested and explored. For example, the NilmDB command line

tools will work, and hardware components such as wireless cards can be tested to ensure compatibility with this version of NilmDBuntu. Note, however, that data capture is not run automatically on the live system, since there is nowhere to store the data.

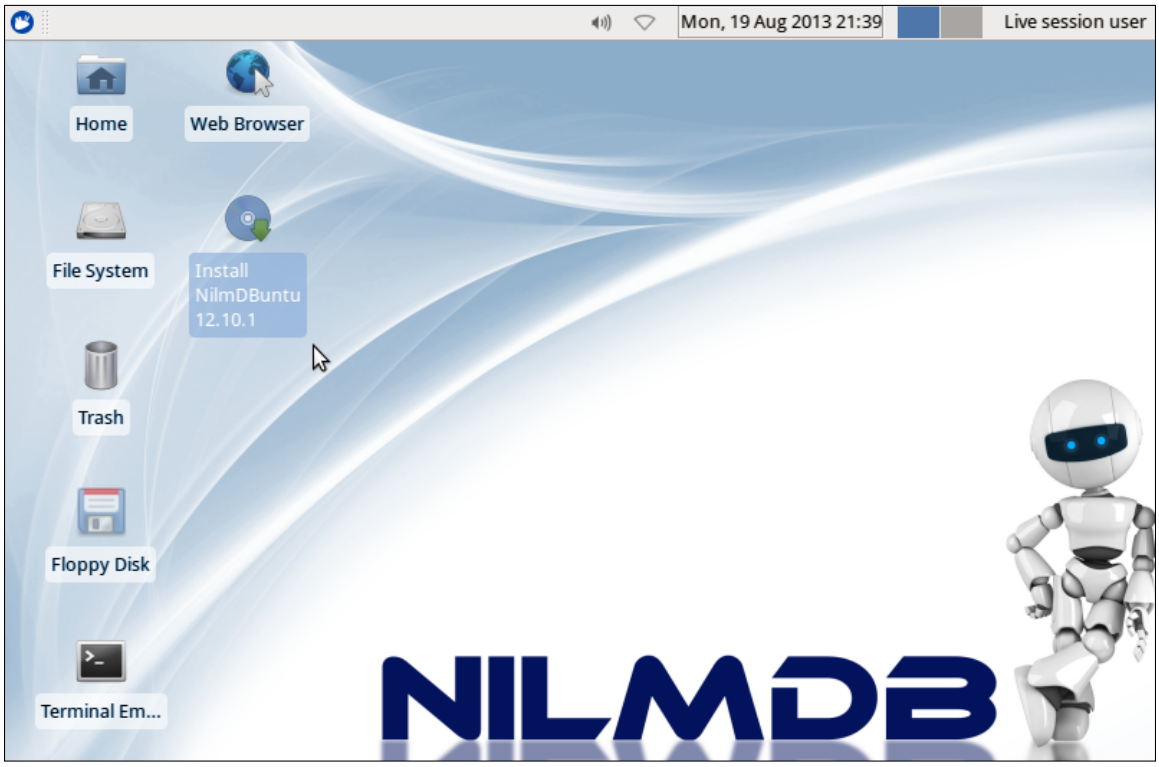


Figure A-2: NilmDBuntu live image, with installer.

To install the image to the hard drive, double-click on the "Install NilmDBuntu" icon on the desktop. This will launch the standard Xubuntu installer, which walks through the installation steps such as partitioning the hard drive, setting up an initial user, etc. In general, the defaults are acceptable and recommended, and the only information that needs to be provided is a password for the initial user.

When the install completes, the installation media can be removed, and the system should boot directly into NilmDBuntu.

A.3 Configuration

The NilmDBuntu install is largely pre-configured for a "typical" NILM use case. The remainder of this appendix discusses the necessary hardware and software setup to complete this configuration and test it.

A.3.1 System time

NilmDB relies on correct system time. If the system clock is not correct, fix it by running a command in the terminal window like:

```
sudo date -s "2013/09/01 14:40"
```

If asked for a password when running "sudo", use the password for the user created during NilmDBuntu installation. Note that the screensaver might immediately kick on if the time changed significantly, and the time shown in the task bar might take a minute to update.

A.3.2 LabJack IP Address Setup

The default configuration assumes a LabJack UE9 Ethernet-based capture card. More information about the LabJack UE9, including purchasing info, can be found at `http://labjack.com/ue9`.

The UE9 must be given an IP address with which to communicate with the NilmDBuntu system. The default IP address is `192.168.1.209` on subnet `255.255.255.0`, although some UE9 devices out of the box use an alternate configuration and request an IP address via DHCP. To reset a UE9 to the default `192.168.1.209`:

1. Power off and disconnect everything from the UE9.
2. Connect a short between pins `FIO2` and `SCL`.
3. Connect power to the UE9.

4. Wait 10 seconds.

5. Disconnect power from the UE9.

6. Remove the short between pins `FIO2` and `SCL`.

The UE9 will then use `192.168.1.209` the next time it is powered up.

Note that the `192.168.1.*` subnet is commonly used by smaller LANs, such as home networking equipment. If the NilmDBuntu system is to be connected to such a network using a second network interface, the IP address of the UE9 should be changed, so that the two network interfaces do not conflict. A suitable alternative to `192.168.1.*` is `172.17.1.*`. Customizing the UE9 IP address can be done most easily on a Windows machine, by downloading and running the "Windows Installer" package from `http://labjack.com/support/ue9`, and using the newly-installed `LJControlPanel` application to change the UE9 configuration. These tools are documented at the LabJack website.

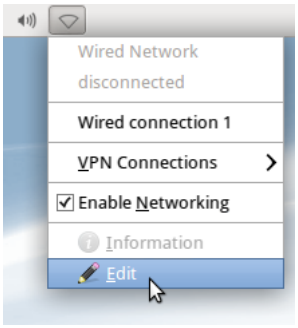
A.3.3 NilmDBuntu Network Configuration

The following steps will configure the NilmDBuntu system to have a static IP address that is on the same subnet as the UE9, so that the two can communicate. The following addresses will be assumed, and should be changed as necessary if a custom network setup is being used:

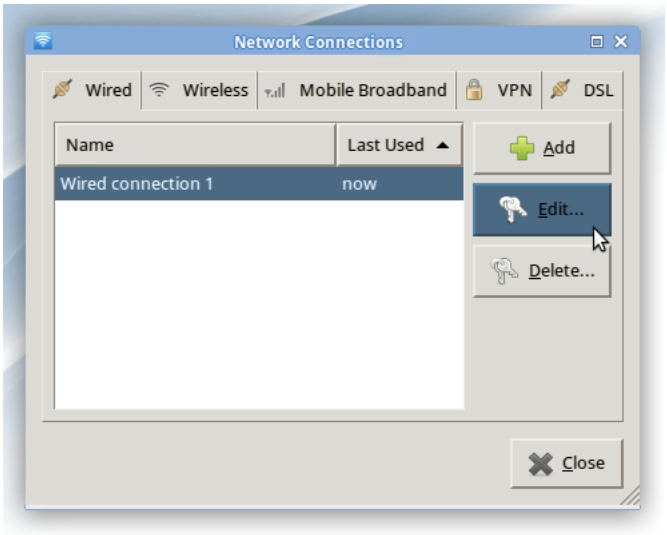
Address	**Description**
`192.168.1.209`	LabJack UE9 IP address.
`255.255.255.0`	Subnet mask for the network.
`192.168.1.200`	Computer IP address, chosen to be different from, but on the same subnet as, the UE9.
`192.168.1.1`	Address of a "gateway" within this network. Not used, but needed for configuration. The computer IP with the last octet set to 1 should be sufficient.

Network configuration can be done through Network Manager as follows:

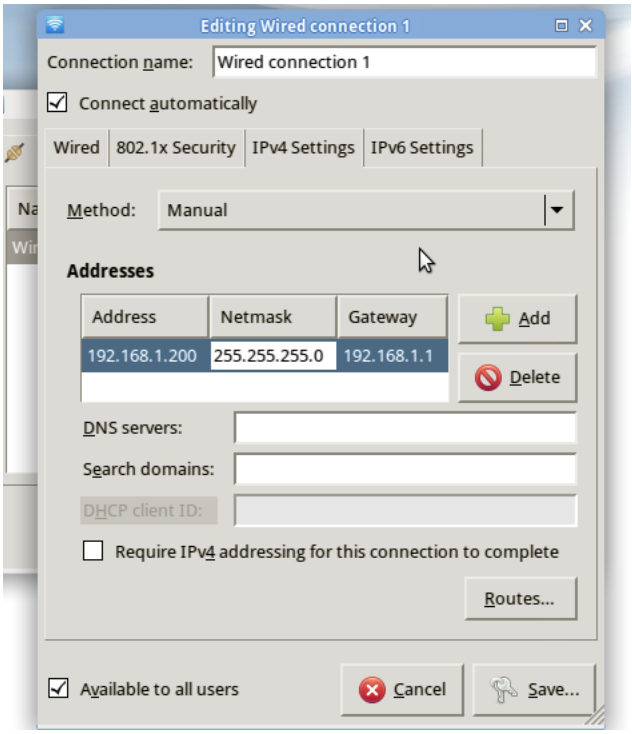
1. Left-click the Network Manager icon in the task bar and select “Edit”.



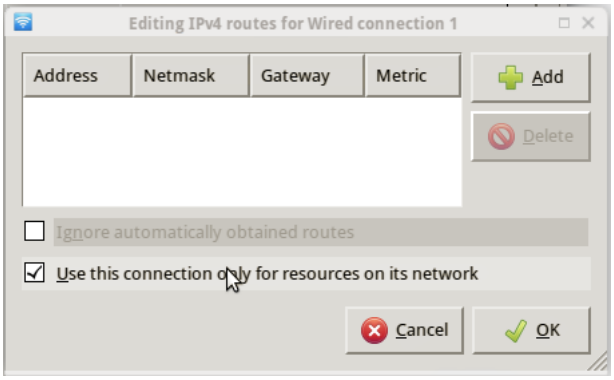
2. The existing network card in the computer should already have a connection associated with it; this is the default “DHCP” configuration that Network Manager sets up out of the box. Select that connection, and click “Edit” again.



3. In the “IPv4 Settings” tab, change the method to “Manual”. Click under “Address” and fill in the computer IP, subnet mask, and gateway address. Do not close this dialog box yet.



4. Click the “Routes...” button, and select the option “Use this connection only for resources on its network”. This essentially disables the gateway, so that this particular network interface is only used for the UE9.



5. Click “OK”, “Save”, and “Close” to apply the settings.

The network configuration can be tested by connecting the UE9, opening a terminal window, and pinging the UE9 by running

```
ping 192.168.1.209
```

which should output something like

```
PING 192.168.1.209 (192.168.1.209) 56(84) bytes of data.
64 bytes from 192.168.1.209: icmp_req=1 ttl=64 time=0.093 ms
64 bytes from 192.168.1.209: icmp_req=2 ttl=64 time=0.073 ms
64 bytes from 192.168.1.209: icmp_req=3 ttl=64 time=0.109 ms
```

Use `Ctrl-C` to kill the `ping` program.

A.3.4 NilmDB configuration overview

NilmDBuntu comes pre-configured to run the database, acquisition, processing, and cleanup as the "`nilmdb`" user. The files in `/home/nilmdb` control the configuration, similar to the setup described in Section 3.1.4. A brief description of each of the relevant files follows:

Cron entries:

The `nilmdb` user's `crontab` starts data capture and processing every 5 minutes. This file is not edited directly, but can be edited by executing `crontab -e`.

`capture.sh`:

Executed by `cron`. This script starts data acquisition, if not already running, from a connected LabJack UE9. It captures six channels at 8 KHz, which are expected to correspond to the 3-phase current and voltage signals IφA, IφB, IφC, VφA, VφB, and VφC. The captured data is written to the NilmDB stream `/data/raw`.

`process.sh`:

Executed by `cron`. This script runs the sinefit spectral envelope preprocessor

and creates output streams `/data/sinefit`, `/data/prep-a`, `/data/prep-b`, and `/data/prep-c`. It also creates decimations for plottable streams, and runs the stream cleanup.`nilm-cleanup`.

`cleanup.cfg`:

Configuration for stream cleanup. By default, two weeks of raw data and one year of preprocessed data are saved. This is expected to take up approximately 900 gigabytes of space.

A.3.5 NilmDB configuration customization

The configuration will need to be updated if the UE9 IP address has changed, or if any other changes from the default setup are desired. To change any of the configuration, first switch to the `nilmdb` user by executing the following in a terminal window:

```
sudo -i -u nilmdb
```

The command `whoami` can be run at any time to see the current username, and `exit` will return to the normal user. Note that only the normal user has a password that can be used to run "sudo".

A.3.5.1 Capture

The first step is to edit `capture.sh`, which can be done with any text editor. Here, `emacs` could be replaced with `nano`, which is simpler to use.

```
emacs capture.sh
```

The default contents of the file are as follows:

```
#!/bin/bash -e
nilm-pipewatch --daemon --lock "/tmp/nilmdb-capture.lock" --timeout 30 \
  "ethstream -a 192.168.1.209 -n 6 -r 8000" \
  "nilm-insert -m 10 -r 8000 --live /data/raw"
```

The `192.168.1.209` address should be changed to the LabJack UE9 IP address. The data rates (both instances of "`-r 8000`") and the number of captured channels ("`-n 6`") could also be changed, if needed.

When changing the capture configuration, any existing capture process should be stopped, so that the `cron` scheduler can start a new one the next time it runs `capture.sh`. To do this, run:

```
killall ethstream
```

The new process should be started by `cron` within 5 minutes. To see whether it is running, run

```
ps axuw
```

and look for "`ethstream`" and "`nilm-pipewatch`" processes.

A.3.5.2 Processing

Similar customization can be done to the processing steps. Edit the `process.sh` file with:

```
emacs process.sh
```

The following are the default steps in the file:

```
# Perform desired processing steps
nilm-sinefit -c 4 /data/raw /data/sinefit
nilm-prep -c 1 -r 0   /data/raw /data/sinefit /data/prep-a
nilm-prep -c 2 -r 120 /data/raw /data/sinefit /data/prep-b
nilm-prep -c 3 -r 240 /data/raw /data/sinefit /data/prep-c
nilm-decimate-auto /data/raw "/data/prep*"
nilm-cleanup --yes /home/nilmdb/cleanup.cfg
```

The `nilm-sinefit` line performs sinusoid fitting on the 4th column (channel) of the raw data stream, which is expected to be a voltage. Adjust the "`-c 4`" to change

it. Similarly, the `nilm-prep` channels might need to be reordered, depending on the phase ordering. This is easiest to do after the system is fully up and running, once the preprocessor output can be seen more easily.

No specific actions are required after editing `process.sh`; the new version will be run automatically when `cron` executes it on its 5-minute schedule.

A.3.5.3 Cleanup

The default cleanup configuration file requires about 900 gigabytes of free disk space. If the system hard drive is smaller than this, or if it is larger and more data should be retained, edit `cleanup.cfg` by running

```
emacs cleanup.cfg
```

The default configuration is to keep raw data for two weeks, and preprocessed and sinefit data for one year:

```
[/data/raw]
keep = 2w

[/data/sinefit]
keep = 1y
decimated = false

[/data/prep-*]
keep = 1y
```

After editing this file, an estimate of how much space will be required can be seen by running:

```
nilm-cleanup --estimate cleanup.cfg
```

Note that this command can only estimate the space once enough data has been collected and processed, because it needs to be able to estimate the data rate. Generally, waiting about 10 minutes after setting up `capture.sh` and `process.sh` will suffice. Al-

ternately, configuration lines like "`rate = 8000`" and "`rate = 60`" could be added to the raw and prep/sinefit configurations, respectively, to force a particular rate to be used for the calculation. See the documentation of `nilm-cleanup` in Section 3.4.2.1 for details.

A.4 Testing and Use

The following is an example of how to test and use the running NilmDBuntu system. Note that the `nilmdb` user should only be used for configuration; other interaction with the database, like the testing here, should be done as the normal user. Use `exit` to return to the normal user, if needed, and switch to the normal user's home directory with `cd`.

A.4.1 Verifying Operation

Proper operation of the NilmDB server can be verified by running

```
nilmtool info
```

which should print a message like

```
Client version: 1.9.7
Server version: 1.9.7
Server URL: http://localhost/nilmdb/
Server database path: /home/nilmdb/db
Server disk space used by NilmDB: 126 kiB
Server disk space used by other: 24.2 MiB
Server disk space reserved: 191 kiB
Server disk space free: 1923.4 GiB
```

The streams in the database can be listed with

```
nilmtool list -n
```

The "`-n`" option excludes decimated streams from the output. This should show:

```
/data/prep-a
/data/prep-b
/data/prep-c
/data/raw
/data/sinefit
```

More detail can be seen by running:

```
nilmtool list -n -E
```

This output will include the "extent" of all data in the stream, showing the earliest and latest intervals available. The output will be similar to:

```
/data/prep-a
  interval extents: Sun, 01 Sep 2013 14:40:06.000000 -0400 -> ↩ (continued)
                    Sun, 01 Sep 2013 14:55:01.000000 -0400
        total data: 54042 rows, 912.173456 seconds
/data/prep-b
  interval extents: Sun, 01 Sep 2013 14:40:06.000000 -0400 -> ↩ (continued)
                    Sun, 01 Sep 2013 14:55:01.000000 -0400
        total data: 54042 rows, 912.173456 seconds
/data/prep-c
  interval extents: Sun, 01 Sep 2013 14:40:06.000000 -0400 -> ↩ (continued)
                    Sun, 01 Sep 2013 14:55:01.000000 -0400
        total data: 54042 rows, 912.173456 seconds
/data/raw
  interval extents: Sun, 01 Sep 2013 14:40:06.000000 -0400 -> ↩ (continued)
                    Sun, 01 Sep 2013 15:03:01.000000 -0400
        total data: 11045838 rows, 1380.189129 seconds
/data/sinefit
  interval extents: Sun, 01 Sep 2013 14:40:06.000000 -0400 -> ↩ (continued)
                    Sun, 01 Sep 2013 14:58:01.000000 -0400
        total data: 56042 rows, 1091.445311 seconds
```

Waiting a few seconds and running this command again should show the extent growing. For the `/data/raw` stream, it will update fairly regularly, while the other streams will only be updated every five minutes when the processing script runs.

A.4.2 Extracting Data

A small sample of raw data can easily be extracted. First, choose a point in time that is known to be present in the `data/raw`, based on the extents output above. Alternately, the command `nilmtool intervals /data/raw` will show all available data intervals, and a time range can be chosen from one of them. Here, assume we want to extract one second of data on September 1st at 14:45. The following command will extract the data and write it to a file, `out.dat`:

```
nilmtool extract -s "20130901 14:45:00" -e "20130901 14:45:01" /data/raw > out.dat
```

A wide variety of date formats can be used; for example, a start time of "Sun, 01 Sep 2013 14:45:00.000000" would have worked.

A.4.3 Plotting Data

NilmDBuntu includes Octave, which is similar to Matlab. To plot the extracted raw data, first load Octave by running:

```
octave
```

and then use, for example, the following Octave commands:

```
load out.dat
t=1:200;
figure(1); plot(out(t,1), out(t,5:7)); title('Voltages')
figure(2); plot(out(t,1), out(t,2:4)); title('Currents')
```

A.5 NILM Manager

The NilmDBuntu system is now fully ready. If a NILM Manager instance is available and can reach the NilmDBuntu system's IP address, NILM Manager can now be configured to point to the NilmDB server at `http://IP/nilmdb`, and to the NilmRun server at `http://IP/nilmrun`.

A.6 NilmDBuntu Source Code

The NilmDBuntu system image is created by scripts that automatically extract, customize, and repack the Xubuntu software distribution. These scripts are included below, and can be used as a baseline for either automatically or manually adding NilmDB software to other images.

Listing A-1: `extractiso.sh`: Extract original Xubuntu desktop image. The live image is unpacked into directories according to the configuration file.

Git repository: `https://git.jim.sh/jim/lees/nilmdbuntu.git`
Filename: `extractiso.sh`
Revision: `ebb3db4ba2d49abc85672f53b4852a2fe0ef7fdf`

```
#!/bin/bash

# build the iso
. config || exit 0

if [ "$1" != "ok" ]; then
    if [ -e ${ISO} ] || [ -e ${FS} ]; then
        echo "remove \"${ISO}\" and \"${FS}\" dirs first,"
        echo "or pass \"ok\" as an argument to remove them"
        exit 0
    fi
fi

set -x
set -e

# download it if it doesn't exist
if ! [ -e ${ORIG} ] ; then
    mkdir -p $(dirname ${ORIG})
    wget -O "${ORIG}" "${ORIGURL}"
fi

# mount it
sudo umount ${MNT} || true
sudo rm -rf ${MNT} ${ISO} ${FS}
sudo mkdir ${MNT}
sudo mount -o loop,ro "$ORIG" ${MNT}

# copy data
sudo mkdir ${ISO}
sudo rsync --exclude=/casper/filesystem.squashfs -a ${MNT}/ ${ISO}
sudo chown -R ${USER} ${ISO}
chmod -R u+w ${ISO}

# copy squashfs
sudo unsquashfs -d ${FS} ${MNT}/casper/filesystem.squashfs

sudo umount ${MNT}
```

Listing A-2: `customize.sh`: Apply NilmDBuntu customizations to the installation disk image.

Git repository: `https://git.jim.sh/jim/lees/nilmdbuntu.git`
Filename: `customize.sh`
Revision: `ebb3db4ba2d49abc85672f53b4852a2fe0ef7fdf`

```
#!/bin/bash

. config || exit 0
set -e
set -x

# Customize the outer ISO image
sed -i -s -e \
    "s/DISKNAME.*/DISKNAME  NilmDBuntu $VERSION by Jim Paris/" \
    ${ISO}/README.diskdefines

# The .disk/info file is important -- it's used by ubiquity to extract
# out the distro name in dialogs, and I think casper might use it too
echo "NilmDBuntu $VERSION by Jim Paris" > ${ISO}/.disk/info

# Set up preseed file
cp ${ISO}/preseed/xubuntu.seed ${ISO}/preseed/nilmdbuntu.seed
cat >> ${ISO}/preseed/nilmdbuntu.seed <<"EOF"
# Commands to run on successful install:
# - Regenerate SSH host keys
# - Remove "quiet splash" from grub command line
# Ubiquity doesn't actually use preseed/late_command, so we do it
# in ubiquity/success_command (which needs things mounted)
ubiquity ubiquity/success_command string \
    echo "success_command running" >/target/var/log/installer/postinst.log; \
    for i in /dev /dev/pts /dev/shm /sys /sys/kernel/security /proc /cdrom; do \
        mount --bind $i /target/$i; done; \
    sed -i -e 's/quiet splash//g' /target/etc/default/grub; \
    chroot /target update-grub \
        >>/target/var/log/installer/postinst.log 2>&1; \
    rm -f /target/etc/ssh/ssh_host_*; \
    chroot /target dpkg-reconfigure openssh-server \
        >>/target/var/log/installer/postinst.log 2>&1; \
    for i in /dev/pts /dev/shm /dev /sys/kernel/security /sys /proc /cdrom; do \
        umount /target/$i; done; \
    echo "success_command done" >>/target/var/log/installer/postinst.log

# Default user.  Ubiquity should let them change this
d-i passwd/user-fullname string NILM User
d-i passwd/username string nilm
#d-i passwd/user-password password nilm
#d-i passwd/user-password-again password nilm
d-i user-setup/allow-password-weak boolean true
d-i passwd/auto-login boolean true
EOF

# Set up isolinux how we want by editing its config
TRY="Boot ^NilmDBuntu ${VERSION}"
cp splash.png ${ISO}/isolinux/splash.png
perl -n -i \
    -e '$n=1 if /^label/; $n=2 if /^label live-install/; next if $n==2;' \
    -e "s/menu label.*(Try|NilmDBuntu).*/menu label $TRY/g;" \
    -e "s,preseed/.*[.]seed,preseed/nilmdbuntu.seed,g;" \
    -e "s/ quiet splash//g;" \
    -e "print;" \
    ${ISO}/isolinux/txt.cfg
sed -i -s -e "s/^ui gfxboot/# ui gfxboot/g;" ${ISO}/isolinux/isolinux.cfg

# Set up grub similarly
```

```
TRY="Boot NilmDBuntu ${VERSION}"
perl -n -i \
    -e "next if /menuentry \"Install/../^}$/;" \
    -e "next if /menuentry \"OEM install/../^}$/;" \
    -e "s/menuentry \"(Try|NilmDBuntu).*\" {/menuentry \"${TRY}\" {/g;" \
    -e "s,preseed/.*[.]seed,preseed/nilmdbuntu.seed,g;" \
    -e "s/ quiet splash//g;" \
    -e "print;" \
    ${ISO}/boot/grub/{grub,loopback}.cfg

if ! [ "$1" == "skip" ] ; then

# Run the customize-inner.sh script inside the chroot
sudo cp nilmdbuntu.png ${FS}/usr/share/xfce4/backdrops/nilmdbuntu.png
sudo cp customize-inner.sh ${FS}/root/customize-inner.sh
sudo chmod +x ${FS}/root/customize-inner.sh
./enter.sh "cd /root ; ./customize-inner.sh"

fi
```

Listing A-3: `customize-inner.sh`: Apply NilmDBuntu customizations to the embedded system image. Installs NilmDB and related tools, and customizes the installed system image.

Git repository: https://git.jim.sh/jim/lees/nilmdbuntu.git
Filename: customize-inner.sh
Revision: ebb3db4ba2d49abc85672f53b4852a2fe0ef7fdf

```
#!/bin/bash

if [ "$IN_CHROOT" != "1" ] ; then
    echo This is supposed to run inside the chroot, oops
    exit 1
fi

set -e
set -x

try_install() {
    # try to install packages, but ignore failure
    for pkg in "$@"; do
        apt-get -y install "$pkg" || true
    done
}

# Set up live username and hostname
cat >/etc/casper.conf <<"EOF"
export USERNAME="ubuntu"
export USERFULLNAME="Live session user"
export HOST="nilmdb"
export BUILD_SYSTEM="Ubuntu"
export FLAVOUR="NilmDBuntu"
EOF

# Upgrade packages, remove old kernels
apt-get update
# in 13.04, doing upgrade & dist-upgrade together tries to install 2 kernels
# at the same time, which breaks for some reason.  Also, try the upgrade
# multiple times since that can help
apt-get -y upgrade || apt-get -y upgrade || true
apt-get -y dist-upgrade || apt-get -y dist-upgrade || true
apt-get -y --purge autoremove
for VER in $(ls --sort=version /lib/modules/ | head -n -1) ; do
```

```
    apt-get -y --purge remove ".*$VER.*"
done

# Disable upgrade popups
sed -i -s -e 's/Prompt=normal/Prompt=never/g' \
    /etc/update-manager/release-upgrades || true

# some stuff we need from Ubuntu
try_install \
    wbritish \
    thunderbird-locale-en-us

# Set up & install postfix for local mail delivery
debconf-set-selections <<"EOF"
postfix postfix/mailname string localdomain
postfix postfix/main_mailer_type select Local only
EOF
apt-get -y install postfix

# install nilmdb things
apt-get -y install \
    python2.7 \
    python2.7-dev \
    python-setuptools \
    python-pip \
    cython \
    git \
    build-essential \
    python-cherrypy3 \
    python-decorator \
    python-simplejson \
    python-requests \
    python-dateutil \
    python-tz \
    python-progressbar \
    python-psutil \
    python-numpy \
    python-nose \
    python-coverage \
    apache2 \
    libapache2-mod-wsgi \
    python-scipy \
    python-daemon

# install other useful but optional stuff
try_install \
    emacs-goodies-el \
    emacs23-nox \
    octave \
    octave-signal \
    octave-missing-functions \
    gnuplot \
    curl \
    gddrescue \
    help2man \
    luatex \
    pgf \
    moreutils \
    ntfsprogs \
    subversion \
    dlocate \
    ack-grep \
    mutt \
    python-matplotlib \
    ipython \
    openvpn \
    network-manager-openvpn-gnome \
```

```
    openssl

# required
apt-get -y install \
    openssh-server

# Set up timezone to America/New_York for the live CD
echo America/New_York > /etc/timezone
dpkg-reconfigure -f noninteractive tzdata

# Create nilmdb user to run the database
adduser --system --group --shell /bin/bash --disabled-password nilmdb
cp -rv /etc/skel/.??* /home/nilmdb
chown -R nilmdb:nilmdb /home/nilmdb

# Create WSGI scripts
cat > /home/nilmdb/nilmdb.wsgi <<"EOF"
import nilmdb.server
application = nilmdb.server.wsgi_application("/home/nilmdb/db","/nilmdb")
EOF
cat > /home/nilmdb/nilmrun.wsgi <<"EOF"
import nilmrun.server
application = nilmrun.server.wsgi_application("/nilmrun")
EOF

# Create apache config by hacking up the default one.  Might be a better way
# to do this, and it'll probably break on apache 2.4, but...
DEF=/etc/apache2/sites-available/default
perl -ne 'print unless /## NilmDB start/../## NilmDB end/' $DEF > $DEF.orig
perl -ne 'print unless m-^[^#]*</VirtualHost>-..1' $DEF.orig > $DEF
cat >>$DEF <<"EOF"
    ## NilmDB start
    WSGIScriptAlias /nilmdb /home/nilmdb/nilmdb.wsgi
    WSGIDaemonProcess nilmdb-procgroup threads=32 user=nilmdb group=nilmdb
    <Location /nilmdb>
        WSGIProcessGroup nilmdb-procgroup
        WSGIApplicationGroup nilmdb-appgroup
    </Location>

    WSGIScriptAlias /nilmrun /home/nilmdb/nilmrun.wsgi
    WSGIDaemonProcess nilmrun-procgroup threads=32 user=nilmdb group=nilmdb
    <Location /nilmrun>
        WSGIProcessGroup nilmrun-procgroup
        WSGIApplicationGroup nilmrun-appgroup
    </Location>
        ## NilmDB end
EOF
perl -ne 'print if m-^[^#]*</VirtualHost>-..1' $DEF.orig >> $DEF

# Create nilmdb capture, processing, and cleanup files
cat > /home/nilmdb/capture.sh <<"EOF"
#!/bin/bash -e

# Don't run capture if we're running off a live CD
if grep -q boot=casper /proc/cmdline ; then
    echo "Skipping capture, because this is a live CD."
    exit 0
fi

echo "Starting capture in background..."
nilm-pipewatch --daemon --lock "/tmp/nilmdb-capture.lock" --timeout 30 \
    "ethstream -a 192.168.1.209 -n 6 -r 8000" \
    "nilm-insert -m 10 -r 8000 --live /data/raw"
EOF
cat > /home/nilmdb/process.sh <<"EOF"
#!/bin/bash -e

```

```
# Ensure only one copy of this code runs at a time:
LOCKFILE="/tmp/nilmdb-process.lock"
exec 99>"$LOCKFILE"
flock -n -x 99 || exit 0
trap 'rm -f "$LOCKFILE"' 0

nilm-sinefit -c 4 /data/raw /data/sinefit
nilm-prep -c 1 -r 0 /data/raw /data/sinefit /data/prep-a
nilm-prep -c 2 -r 120 /data/raw /data/sinefit /data/prep-b
nilm-prep -c 3 -r 240 /data/raw /data/sinefit /data/prep-c
nilm-decimate-auto /data/raw "/data/prep*"
nilm-cleanup --yes /home/nilmdb/cleanup.cfg
EOF
cat > /home/nilmdb/cleanup.cfg <<"EOF"
[/data/prep-*]
keep = 1y

[/data/raw]
keep = 2w

[/data/sinefit]
keep = 1y
decimated = false
EOF

# Set up crontab
cat > /home/nilmdb/crontab <<"EOF"
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Run capture and processing scripts every 5 minutes
*/5 * * * * chronic /home/nilmdb/capture.sh
*/5 * * * * chronic /home/nilmdb/process.sh

# Try to run nilmdb-fsck on boot.  It should hopefully run before
# apache opens the database, and apache will return errors to clients
# until nilmdb-fsck is done.
@reboot chronic nilmdb-fsck --fix --no-data /home/nilmdb/db
EOF
crontab -u nilmdb /home/nilmdb/crontab

# Fix permissions
chown -R nilmdb:nilmdb /home/nilmdb
chmod +x /home/nilmdb/{capture,process}.sh

# Fetch and build everything.  Put it in the nilmdb dir
echo "machine git.jim.sh login nilm password nilm" > /home/nilmdb/.netrc
GIT=https://git.jim.sh/jim/lees
rm -rf /home/nilmdb/git
mkdir /home/nilmdb/git
chown nilmdb:nilmdb /home/nilmdb/.netrc /home/nilmdb/git
REPOS="nilmdb nilmtools nilmrun ethstream"

# check it out as nilmdb, so the .netrc gets used
for repo in $REPOS; do
    sudo -i -u nilmdb git clone $GIT/$repo.git git/$repo
done

# build as root, because we need to do that for the install
for repo in $REPOS; do
    make -C /home/nilmdb/git/$repo install
done

# fix up all permissions in git dir, so nilmdb user can play with it later
chown -R nilmdb:nilmdb /home/nilmdb/git

# Create the initial database and streams by running the standalone
```

```
# server as nilmdb, making the right nilmtool calls, and killing it.
sudo -i -u nilmdb nilmdb-server -a 127.0.0.1 -p 18646 -d /home/nilmdb/db &
SERVERPID=$!
trap "kill -9 $SERVERPID" 0
for i in $(seq 1 120) ; do
    sleep 1
    echo waiting for nilmdb to start $i
    if nilmtool -u http://127.0.0.1:18646/ info ; then
        break
    fi
done
nilmtool -u http://127.0.0.1:18646/ destroy -R "/data/*" || true
nilmtool -u http://127.0.0.1:18646/ create /data/raw uint16_6
nilmtool -u http://127.0.0.1:18646/ create /data/sinefit float32_3
nilmtool -u http://127.0.0.1:18646/ create /data/prep-a float32_8
nilmtool -u http://127.0.0.1:18646/ create /data/prep-b float32_8
nilmtool -u http://127.0.0.1:18646/ create /data/prep-c float32_8
kill $SERVERPID
wait
trap "" 0

# Put some default desktop shortcuts in place
DESKTOP=/etc/skel/Desktop
mkdir -p $DESKTOP
cp /usr/share/applications/exo-terminal-emulator.desktop $DESKTOP || true
cp /usr/share/applications/exo-web-browser.desktop $DESKTOP || true
chmod +x $DESKTOP/*  # needs to be executable for 13.04+

# XFCE / theme customizations
if [ -d /usr/share/themes/Clearlooks ] && \
    [ -d /usr/share/icons/elementary-xfce ] ; then
    cat > /usr/share/gconf/defaults/88_nilmdbuntu-settings <<"EOF"
/desktop/gnome/interface/gtk_theme "Clearlooks"
/desktop/gnome/interface/icon_theme "elementary-xfce"
EOF
    update-gconf-defaults
fi

XML=/etc/xdg/xdg-xubuntu/xfce4/xfconf/xfce-perchannel-xml
BG=/usr/share/xfce4/backdrops
mkdir -p $XML
cat >$XML/xfce4-desktop.xml <<"EOF"
<?xml version="1.0" encoding="UTF-8"?>

<channel name="xfce4-desktop" version="1.0">
  <property name="desktop-icons" type="empty">
    <property name="style" type="int" value="2"/>
    <property name="file-icons" type="empty">
      <property name="show-home" type="bool" value="true"/>
      <property name="show-filesystem" type="bool" value="true"/>
      <property name="show-removable" type="bool" value="true"/>
      <property name="show-trash" type="bool" value="true"/>
    </property>
  </property>
  <property name="backdrop" type="empty">
    <property name="screen0" type="empty">
      <property name="monitor0" type="empty">
        <property name="image-path" type="string"
         value="/usr/share/xfce4/backdrops/nilmdbuntu.png"/>
        <property name="image-show" type="bool" value="true"/>
        <property name="image-style" type="int" value="4"/>
        <property name="color-style" type="int" value="0"/>
        <property name="color1" type="array">
          <value type="uint" value="0"/>
          <value type="uint" value="0"/>
          <value type="uint" value="0"/>
          <value type="uint" value="65535"/>
```

```
        </property>
      </property>
      <property name="monitor1" type="empty">
        <property name="image-path" type="string"
         value="/usr/share/xfce4/backdrops/nilmdbuntu.png"/>
        <property name="image-show" type="bool" value="true"/>
        <property name="image-style" type="int" value="4"/>
        <property name="color-style" type="int" value="0"/>
        <property name="color1" type="array">
          <value type="uint" value="0"/>
          <value type="uint" value="0"/>
          <value type="uint" value="0"/>
          <value type="uint" value="65535"/>
        </property>
      </property>
    </property>
  </property>
</channel>
EOF
sed -i -s -e 's/Greybird/Default/g' $XML/xfwm4.xml || true
sed -i -s -e 's/Greybird/Clearlooks/g' $XML/xsettings.xml || true
sed -i -s -e \
    's/elementary-xfce-dark/elementary-xfce/g' $XML/xsettings.xml || true

# Firefox defaults
cat >/etc/firefox/syspref.js <<"EOF"
pref("browser.startup.homepage", "http://nilmdb.com/");
EOF
cat >/etc/xul-ext/homepage.properties <<"EOF"
browser.startup.homepage=http://nilmdb.com/
EOF
cat >/etc/xul-ext/ubufox.js <<"EOF"
pref("browser.startup.homepage", "file:/etc/xul-ext/homepage.properties");
EOF
```

Listing A-4: `enter.sh`: Mount and enter the system image. Used by `customize-inner.sh`.

Git repository: `https://git.jim.sh/jim/lees/nilmdbuntu.git`
Filename: `enter.sh`
Revision: `ebb3db4ba2d49abc85672f53b4852a2fe0ef7fdf`

```
#!/bin/bash

# make sure this was run as root
if [ $UID -ne 0 ] ; then
    echo "Need to be root; trying sudo"
    exec sudo env BUILD_CONFIG=$BUILD_CONFIG $0 "$@"
fi

# enter the chroot and run the command (if supplied) or a shell
. config || exit 0

FAILED=0
run() {
    echo "+" "$1"
    chroot ${FS} env -i \
        HOME=/root \
        PATH=/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin \
        TERM=$TERM \
        IN_CHROOT=1 \
        bash -c "$1"
    RET=$?
```

```
    if [ $RET -ne 0 ] && [ "$1" != "exec bash" ] ; then
        printf "%s\n" "----------- WARNING: failed with exit code $RET"
        FAILED=$RET
        sleep 5
    fi
}

set -e
mount -t proc none ${FS}/proc
mount -t sysfs none ${FS}/sys
mount -t devpts none ${FS}/dev/pts

run "echo 'nameserver 8.8.8.8' > /etc/resolv.conf"
run "dbus-uuidgen > /var/lib/dbus/machine-id"
run "dpkg-divert --local --rename --add /sbin/initctl"
run "ln -s /bin/true /sbin/initctl"
run "dpkg-divert --local --rename --add /usr/sbin/update-grub"
run "ln -s /bin/true /usr/sbin/update-grub"

set +e
if [ -z "$1" ] ; then
    run "exec bash"
else
    run "$1"
fi
run "apt-get clean"
run "rm /sbin/initctl"
run "dpkg-divert --rename --remove /sbin/initctl"
run "rm /usr/sbin/update-grub"
run "dpkg-divert --rename --remove /usr/sbin/update-grub"
run "rm /var/lib/dbus/machine-id"
run "> /etc/resolv.conf"
run "rm -rf /tmp/* /tmp/.??* /root/.bash_history"

umount ${FS}/dev/pts
umount ${FS}/sys/kernel/security || true
umount ${FS}/sys
umount ${FS}/proc

echo "cleaned up"
if [ $FAILED -ne 0 ] ; then
    exit $FAILED
fi
exit 0
```

Listing A-5: `buildiso.sh`: Build a bootable NilmDBuntu image. The image is built to boot on both legacy BIOS and EFI systems, and can be either burned to a DVD or copied directly to USB media.

Git repository: `https://git.jim.sh/jim/lees/nilmdbuntu.git`
Filename: `buildiso.sh`
Revision: `ebb3db4ba2d49abc85672f53b4852a2fe0ef7fdf`

```
#!/bin/bash

# build the iso
. config || exit 0

set -x
set -e

if ! [ "$1" == "skip" ] ; then

```

```
# copy kernel if changed
if [ ${FS}/initrd.img -nt ${ISO}/casper/initrd.lz ] ; then
    sudo cp ${FS}/vmlinuz ${ISO}/casper/vmlinuz
    sudo sh -c "zcat ${FS}/initrd.img | lzma > ${ISO}/casper/initrd.lz"
fi

# manifests
sudo chmod +w ${ISO}/casper/filesystem.manifest
sudo chroot ${FS} dpkg-query -W --showformat='${Package} ${Version}\n' \
    | sudo tee ${ISO}/casper/filesystem.manifest >/dev/null

# squashfs
sudo rm -f ${ISO}/casper/filesystem.squashfs
sudo mksquashfs ${FS} ${ISO}/casper/filesystem.squashfs
printf $(sudo du -sx --block-size=1 ${FS} | cut -f1) \
    | sudo tee ${ISO}/casper/filesystem.size

fi

# md5sums
sudo rm -f md5sum.txt
sudo find ${ISO} -type f -print0 \
    | sudo xargs -0 md5sum \
    | sed -e "s, ${ISO}, ./," \
    | grep -v isolinux/boot.cat \
    | grep -v isolinux/isolinux.bin \
    | grep -v md5sum.txt \
    | sudo tee ${ISO}/md5sum.txt >/dev/null

sudo chown -R ${USER} ${ISO}

# build CD
xorriso -as mkisofs \
    -D -r -V "NilmDBuntu ${VERSION}" -cache-inodes -J -l \
    -input-charset utf-8 -o ${OUTPUT} \
    -b isolinux/isolinux.bin -c isolinux/boot.cat -no-emul-boot \
    -boot-load-size 4 -boot-info-table \
    -eltorito-alt-boot -e boot/grub/efi.img -no-emul-boot \
    ${ISO}

# fix iso for hybrid booting
isohybrid ${OUTPUT}

set +x
echo "Burn it with:"
echo "  growisofs -dvd-compat -Z /dev/dvd=${OUTPUT}"
echo "or write directly to a USB key"
```

Appendix B

NilmDB Implementation

This appendix contains the complete implementation of NilmDB, including the server, client libraries, and command line management tools. Additional tools and the filter library are included in Appendix E.

B.1 Server

Listing B-1: `nilmdb/scripts/nilmdb_server.py`: Script to spawn the standalone NilmDB server. This starts the NilmDB server using the CherryPy HTTP server, and is typically used for debug or test deployments.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/scripts/nilmdb_server.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  #!/usr/bin/python
2
3  import nilmdb.server
4  import argparse
5  import os
6  import socket
7
8  def main():
9      """Main entry point for the 'nilmdb-server' command line script"""
10
11     parser = argparse.ArgumentParser(
12         description = 'Run the NilmDB server',
13         formatter_class = argparse.ArgumentDefaultsHelpFormatter,
14         version = nilmdb.__version__)
15
16     group = parser.add_argument_group("Standard options")
17     group.add_argument('-a', '--address',
18                        help = 'Only listen on the given address',
19                        default = '0.0.0.0')
20     group.add_argument('-p', '--port', help = 'Listen on the given port',
```

```
                        type = int, default = 12380)
    group.add_argument('-d', '--database', help = 'Database directory',
                        default = "./db")
    group.add_argument('-q', '--quiet', help = 'Silence output',
                        action = 'store_true')
    group.add_argument('-t', '--traceback',
                        help = 'Provide tracebacks in client errors',
                        action = 'store_true', default = False)

    group = parser.add_argument_group("Debug options")
    group.add_argument('-y', '--yappi', help = 'Run under yappi profiler and '
                        'invoke interactive shell afterwards',
                        action = 'store_true')

    args = parser.parse_args()

    # Create database object.  Needs to be serialized before passing
    # to the Server.
    db = nilmdb.utils.serializer_proxy(nilmdb.server.NilmDB)(args.database)

    # Configure the server
    if args.quiet:
        embedded = True
    else:
        embedded = False
    server = nilmdb.server.Server(db,
                                  host = args.address,
                                  port = args.port,
                                  embedded = embedded,
                                  force_traceback = args.traceback)

    # Print info
    if not args.quiet:
        print "Version: %s" % nilmdb.__version__
        print "Database: %s" % (os.path.realpath(args.database))
        if args.address == '0.0.0.0' or args.address == '::':
            host = socket.getfqdn()
        else:
            host = args.address
        print "Server URL: http://%s:%d/" % ( host, args.port)
        print "----"

    # Run it
    if args.yappi:
        print "Running in yappi"
        try:
            import yappi
            yappi.start()
            server.start(blocking = True)
        finally:
            yappi.stop()
            yappi.print_stats(sort_type = yappi.SORTTYPE_TTOT, limit = 50)
            from IPython import embed
            embed(header = "Use the yappi object to explore further, "
                  "quit to exit")
    else:
        server.start(blocking = True)

    # Clean up
    if not args.quiet:
        print "Closing database"
        db.close()

if __name__ == "__main__":
    main()
```

Listing B-2: `nilmdb/server/__init__.py`: Server module initialization. This code handles the automatic building of `.pyx` files through `pyximport`, if Cython is available. Otherwise, precompiled modules, typically generated during the installation process, must be present.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/server/__init__.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
"""nilmdb.server"""

from __future__ import absolute_import

# Try to set up pyximport to automatically rebuild Cython modules.  If
# this doesn't work, it's OK, as long as the modules were built externally.
# (e.g. python setup.py build_ext --inplace)
try: # pragma: no cover
    import Cython
    import distutils.version
    if (distutils.version.LooseVersion(Cython.__version__) <
        distutils.version.LooseVersion("0.17")): # pragma: no cover
        raise ImportError("Cython version too old")
    import pyximport
    pyximport.install(inplace = True, build_in_temp = False)
except (ImportError, TypeError): # pragma: no cover
    pass

from nilmdb.server.nilmdb import NilmDB
from nilmdb.server.server import Server, wsgi_application
from nilmdb.server.errors import NilmDBError, StreamError, OverlapError
```

Listing B-3: `nilmdb/server/nilmdb.py`: Primary NilmDB interface and management class. The `NilmDB` class expects to be executed from a single thread at all times, so it must be wrapped by the serialization proxy in Listing B-40.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/server/nilmdb.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
# -*- coding: utf-8 -*-

"""NilmDB

Object that represents a NILM database file.

Manages both the SQL database and the table storage backend.
"""

# Need absolute_import so that "import nilmdb" won't pull in
# nilmdb.py, but will pull the parent nilmdb module instead.
from __future__ import absolute_import
import nilmdb.utils
from nilmdb.utils.printf import *
from nilmdb.utils.time import timestamp_to_string

from nilmdb.utils.interval import IntervalError
from nilmdb.server.interval import Interval, DBInterval, IntervalSet

from nilmdb.server import bulkdata
```

```
from nilmdb.server.errors import NilmDBError, StreamError, OverlapError

import sqlite3
import os
import errno
import bisect

# Note about performance and transactions:
#
# Committing a transaction in the default sync mode (PRAGMA synchronous=FULL)
# takes about 125msec.  sqlite3 will commit transactions at 3 times:
# 1: explicit con.commit()
# 2: between a series of DML commands and non-DML commands, e.g.
#    after a series of INSERT, SELECT, but before a CREATE TABLE or PRAGMA.
# 3: at the end of an explicit transaction, e.g. "with self.con as con:"
#
# To speed things up, we can set 'PRAGMA synchronous=OFF'.  Or, it
# seems that 'PRAGMA synchronous=NORMAL' and 'PRAGMA journal_mode=WAL'
# give an equivalent speedup more safely.  That is what is used here.
_sql_schema_updates = {
    0: { "next": 1, "sql": """
    -- All streams
    CREATE TABLE streams(
        id INTEGER PRIMARY KEY,     -- stream ID
        path TEXT UNIQUE NOT NULL,  -- path, e.g. '/newton/prep'
        layout TEXT NOT NULL        -- layout name, e.g. float32_8
    );

    -- Individual timestamped ranges in those streams.
    -- For a given start_time and end_time, this tells us that the
    -- data is stored between start_pos and end_pos.
    -- Times are stored as µs since Unix epoch
    -- Positions are opaque: PyTables rows, file offsets, etc.
    --
    -- Note: end_pos points to the row _after_ end_time, so end_pos-1
    -- is the last valid row.
    CREATE TABLE ranges(
        stream_id INTEGER NOT NULL,
        start_time INTEGER NOT NULL,
        end_time INTEGER NOT NULL,
        start_pos INTEGER NOT NULL,
        end_pos INTEGER NOT NULL
    );
    CREATE INDEX _ranges_index ON ranges (stream_id, start_time, end_time);
    """ },

    1: { "next": 3, "sql": """
    -- Generic dictionary-type metadata that can be associated with a stream
    CREATE TABLE metadata(
        stream_id INTEGER NOT NULL,
        key TEXT NOT NULL,
        value TEXT
    );
    """ },

    2: { "error": "old format with floating-point timestamps requires "
         "nilmdb 1.3.1 or older" },

    3: { "next": None },
}

@nilmdb.utils.must_close()
class NilmDB(object):
    verbose = 0

    def __init__(self, basepath, max_results=None,
                 max_removals=None, bulkdata_args=None):
```

```
        """Initialize NilmDB at the given basepath.
        Other arguments are for debugging / testing:

        'max_results' is the max rows to send in a single
        stream_intervals or stream_extract response.

        'max_removals' is the max rows to delete at once
        in stream_move.

        'bulkdata_args' is kwargs for the bulkdata module.
        """
        if bulkdata_args is None:
            bulkdata_args = {}

        # set up path
        self.basepath = os.path.abspath(basepath)

        # Create the database path if it doesn't exist
        try:
            os.makedirs(self.basepath)
        except OSError as e:
            if e.errno != errno.EEXIST: # pragma: no cover
                # (no coverage, because it's hard to trigger this case
                # if tests are run as root)
                raise IOError("can't create tree " + self.basepath)

        # Our data goes inside it
        self.data = bulkdata.BulkData(self.basepath, **bulkdata_args)

        # SQLite database too
        sqlfilename = os.path.join(self.basepath, "data.sql")
        self.con = sqlite3.connect(sqlfilename, check_same_thread = True)
        try:
            self._sql_schema_update()
        except Exception: # pragma: no cover
            self.data.close()
            raise

        # See big comment at top about the performance implications of this
        self.con.execute("PRAGMA synchronous=NORMAL")
        self.con.execute("PRAGMA journal_mode=WAL")

        # Approximate largest number of elements that we want to send
        # in a single reply (for stream_intervals, stream_extract).
        self.max_results = max_results or 16384

        # Remove up to this many rows per call to stream_remove.
        self.max_removals = max_removals or 1048576

    def get_basepath(self):
        return self.basepath

    def close(self):
        if self.con:
            self.con.commit()
            self.con.close()
        self.data.close()

    def _sql_schema_update(self):
        cur = self.con.cursor()
        version = cur.execute("PRAGMA user_version").fetchone()[0]
        oldversion = version

        while True:
            if version not in _sql_schema_updates: # pragma: no cover
                raise Exception(self.basepath + ": unknown database version "
                                + str(version))
```

```
            update = _sql_schema_updates[version]
            if "error" in update: # pragma: no cover
                raise Exception(self.basepath + ": can't use database version "
                                + str(version) + ": " + update["error"])
            if update["next"] is None:
                break
            cur.executescript(update["sql"])
            version = update["next"]
            if self.verbose: # pragma: no cover
                printf("Database schema updated to %d\n", version)

        if version != oldversion:
            with self.con:
                cur.execute("PRAGMA user_version = {v:d}".format(v=version))

    def _check_user_times(self, start, end):
        if start is None:
            start = nilmdb.utils.time.min_timestamp
        if end is None:
            end = nilmdb.utils.time.max_timestamp
        if start >= end:
            raise NilmDBError("start must precede end")
        return (start, end)

    @nilmdb.utils.lru_cache(size = 64)
    def _get_intervals(self, stream_id):
        """
        Return a mutable IntervalSet corresponding to the given stream ID.
        """
        iset = IntervalSet()
        result = self.con.execute("SELECT start_time, end_time, "
                                  "start_pos, end_pos "
                                  "FROM ranges "
                                  "WHERE stream_id=?", (stream_id,))
        try:
            for (start_time, end_time, start_pos, end_pos) in result:
                iset += DBInterval(start_time, end_time,
                                   start_time, end_time,
                                   start_pos, end_pos)
        except IntervalError: # pragma: no cover
            raise NilmDBError("unexpected overlap in ranges table!")

        return iset

    def _sql_interval_insert(self, id, start, end, start_pos, end_pos):
        """Helper that adds interval to the SQL database only"""
        self.con.execute("INSERT INTO ranges "
                         "(stream_id,start_time,end_time,start_pos,end_pos) "
                         "VALUES (?,?,?,?,?)",
                         (id, start, end, start_pos, end_pos))

    def _sql_interval_delete(self, id, start, end, start_pos, end_pos):
        """Helper that removes interval from the SQL database only"""
        self.con.execute("DELETE FROM ranges WHERE "
                         "stream_id=? AND start_time=? AND "
                         "end_time=? AND start_pos=? AND end_pos=?",
                         (id, start, end, start_pos, end_pos))

    def _add_interval(self, stream_id, interval, start_pos, end_pos):
        """
        Add interval to the internal interval cache, and to the database.
        Note: arguments must be ints (not numpy.int64, etc)
        """
        # Load this stream's intervals
        iset = self._get_intervals(stream_id)

        # Check for overlap
```

```
222             if iset.intersects(interval): # pragma: no cover (gets caught earlier)
223                 raise NilmDBError("new interval overlaps existing data")
224 
225             # Check for adjacency.  If there's a stream in the database
226             # that ends exactly when this one starts, and the database
227             # rows match up, we can make one interval that covers the
228             # time range [adjacent.start -> interval.end)
229             # and database rows [ adjacent.start_pos -> end_pos ].
230             # Only do this if the resulting interval isn't too large.
231             max_merged_rows = 8000 * 60 * 60 * 1.05 # 1.05 hours at 8 KHz
232             adjacent = iset.find_end(interval.start)
233             if (adjacent is not None and
234                 start_pos == adjacent.db_endpos and
235                 (end_pos - adjacent.db_startpos) < max_merged_rows):
236                 # First delete the old one, both from our iset and the
237                 # database
238                 iset -= adjacent
239                 self._sql_interval_delete(stream_id,
240                                           adjacent.db_start, adjacent.db_end,
241                                           adjacent.db_startpos, adjacent.db_endpos)
242 
243                 # Now update our interval so the fallthrough add is
244                 # correct.
245                 interval.start = adjacent.start
246                 start_pos = adjacent.db_startpos
247 
248             # Add the new interval to the iset
249             iset.iadd_nocheck(DBInterval(interval.start, interval.end,
250                                          interval.start, interval.end,
251                                          start_pos, end_pos))
252 
253             # Insert into the database
254             self._sql_interval_insert(stream_id, interval.start, interval.end,
255                                       int(start_pos), int(end_pos))
256 
257             self.con.commit()
258 
259         def _remove_interval(self, stream_id, original, remove):
260             """
261             Remove an interval from the internal cache and the database.
262 
263             stream_id: id of stream
264              original: original DBInterval; must be already present in DB
265             to_remove: DBInterval to remove; must be subset of 'original'
266             """
267             # Just return if we have nothing to remove
268             if remove.start == remove.end: # pragma: no cover
269                 return
270 
271             # Load this stream's intervals
272             iset = self._get_intervals(stream_id)
273 
274             # Remove existing interval from the cached set and the database
275             iset -= original
276             self._sql_interval_delete(stream_id,
277                                       original.db_start, original.db_end,
278                                       original.db_startpos, original.db_endpos)
279 
280             # Add back the intervals that would be left over if the
281             # requested interval is removed.  There may be two of them, if
282             # the removed piece was in the middle.
283             def add(iset, start, end, start_pos, end_pos):
284                 iset += DBInterval(start, end, start, end, start_pos, end_pos)
285                 self._sql_interval_insert(stream_id, start, end, start_pos, end_pos)
286 
287             if original.start != remove.start:
288                 # Interval before the removed region
```

```
289             add(iset, original.start, remove.start,
290                 original.db_startpos, remove.db_startpos)
291 
292         if original.end != remove.end:
293             # Interval after the removed region
294             add(iset, remove.end, original.end,
295                 remove.db_endpos, original.db_endpos)
296 
297         # Commit SQL changes
298         self.con.commit()
299 
300         return
301 
302     def stream_list(self, path = None, layout = None, extended = False):
303         """Return list of lists of all streams in the database.
304 
305         If path is specified, include only streams with a path that
306         matches the given string.
307 
308         If layout is specified, include only streams with a layout
309         that matches the given string.
310 
311         If extended = False, returns a list of lists containing
312         the path and layout: [ path, layout ]
313 
314         If extended = True, returns a list of lists containing
315         more information:
316            path
317            layout
318            interval_min (earliest interval start)
319            interval_max (latest interval end)
320            rows         (total number of rows of data)
321            time         (total time covered by this stream, in timestamp units)
322         """
323         params = ()
324         query = "SELECT streams.path, streams.layout"
325         if extended:
326             query += ", min(ranges.start_time), max(ranges.end_time) "
327             query += ", coalesce(sum(ranges.end_pos - ranges.start_pos), 0) "
328             query += ", coalesce(sum(ranges.end_time - ranges.start_time), 0) "
329         query += " FROM streams"
330         if extended:
331             query += " LEFT JOIN ranges ON streams.id = ranges.stream_id"
332         query += " WHERE 1=1"
333         if layout is not None:
334             query += " AND streams.layout=?"
335             params += (layout,)
336         if path is not None:
337             query += " AND streams.path=?"
338             params += (path,)
339         query += " GROUP BY streams.id ORDER BY streams.path"
340         result = self.con.execute(query, params).fetchall()
341         return [ list(x) for x in result ]
342 
343     def stream_intervals(self, path, start = None, end = None, diffpath = None):
344         """
345         List all intervals in 'path' between 'start' and 'end'.  If
346         'diffpath' is not none, list instead the set-difference
347         between the intervals in the two streams; i.e. all interval
348         ranges that are present in 'path' but not 'diffpath'.
349 
350         Returns (intervals, restart) tuple.
351 
352         'intervals' is a list of [start,end] timestamps of all intervals
353         that exist for path, between start and end.
354 
355         'restart', if not None, means that there were too many results
```

```
356             to return in a single request.  The data is complete from the
357             starting timestamp to the point at which it was truncated, and
358             a new request with a start time of 'restart' will fetch the
359             next block of data.
360             """
361             stream_id = self._stream_id(path)
362             intervals = self._get_intervals(stream_id)
363             if diffpath:
364                 diffstream_id = self._stream_id(diffpath)
365                 diffintervals = self._get_intervals(diffstream_id)
366             (start, end) = self._check_user_times(start, end)
367             requested = Interval(start, end)
368             result = []
369             if diffpath:
370                 getter = nilmdb.utils.interval.set_difference(
371                     intervals.intersection(requested),
372                     diffintervals.intersection(requested))
373             else:
374                 getter = intervals.intersection(requested)
375             for n, i in enumerate(getter):
376                 if n >= self.max_results:
377                     restart = i.start
378                     break
379                 result.append([i.start, i.end])
380             else:
381                 restart = None
382             return (result, restart)
383 
384         def stream_create(self, path, layout_name):
385             """Create a new table in the database.
386 
387             path: path to the data (e.g. '/newton/prep').
388             Paths must contain at least two elements, e.g.:
389                /newton/prep
390                /newton/raw
391                /newton/upstairs/prep
392                /newton/upstairs/raw
393 
394             layout_name: string for nilmdb.layout.get_named(), e.g. 'float32_8'
395             """
396             # Create the bulk storage.  Raises ValueError on error, which we
397             # pass along.
398             self.data.create(path, layout_name)
399 
400             # Insert into SQL database once the bulk storage is happy
401             with self.con as con:
402                 con.execute("INSERT INTO streams (path, layout) VALUES (?,?)",
403                             (path, layout_name))
404 
405         def _stream_id(self, path):
406             """Return unique stream ID"""
407             result = self.con.execute("SELECT id FROM streams WHERE path=?",
408                                       (path,)).fetchone()
409             if result is None:
410                 raise StreamError("No stream at path " + path)
411             return result[0]
412 
413         def stream_set_metadata(self, path, data):
414             """Set stream metadata from a dictionary, e.g.
415                { description = 'Downstairs lighting',
416                  v_scaling = 123.45 }
417                This replaces all existing metadata.
418                """
419             stream_id = self._stream_id(path)
420             with self.con as con:
421                 con.execute("DELETE FROM metadata WHERE stream_id=?", (stream_id,))
422                 for key in data:
```

```
423                 if data[key] != '':
424                     con.execute("INSERT INTO metadata VALUES (?, ?, ?)",
425                                 (stream_id, key, data[key]))
426 
427     def stream_get_metadata(self, path):
428         """Return stream metadata as a dictionary."""
429         stream_id = self._stream_id(path)
430         result = self.con.execute("SELECT metadata.key, metadata.value "
431                                   "FROM metadata "
432                                   "WHERE metadata.stream_id=?", (stream_id,))
433         data = {}
434         for (key, value) in result:
435             data[key] = value
436         return data
437 
438     def stream_update_metadata(self, path, newdata):
439         """Update stream metadata from a dictionary"""
440         data = self.stream_get_metadata(path)
441         data.update(newdata)
442         self.stream_set_metadata(path, data)
443 
444     def stream_rename(self, oldpath, newpath):
445         """Rename a stream."""
446         stream_id = self._stream_id(oldpath)
447 
448         # Rename the data
449         self.data.rename(oldpath, newpath)
450 
451         # Rename the stream in the database
452         with self.con as con:
453             con.execute("UPDATE streams SET path=? WHERE id=?",
454                         (newpath, stream_id))
455 
456     def stream_destroy(self, path):
457         """Fully remove a table from the database.  Fails if there are
458         any intervals data present; remove them first.  Metadata is
459         also removed."""
460         stream_id = self._stream_id(path)
461 
462         # Verify that no intervals are present, and clear the cache
463         iset = self._get_intervals(stream_id)
464         if len(iset):
465             raise NilmDBError("all intervals must be removed before "
466                               "destroying a stream")
467         self._get_intervals.cache_remove(self, stream_id)
468 
469         # Delete the bulkdata storage
470         self.data.destroy(path)
471 
472         # Delete metadata, stream, intervals (should be none)
473         with self.con as con:
474             con.execute("DELETE FROM metadata WHERE stream_id=?", (stream_id,))
475             con.execute("DELETE FROM ranges WHERE stream_id=?", (stream_id,))
476             con.execute("DELETE FROM streams WHERE id=?", (stream_id,))
477 
478     def stream_insert(self, path, start, end, data, binary = False):
479         """Insert new data into the database.
480            path: Path at which to add the data
481            start: Starting timestamp
482            end: Ending timestamp
483            data: Textual data, formatted according to the layout of path
484 
485            'binary', if True, means that 'data' is raw binary:
486            little-endian, matching the current table's layout,
487            including the int64 timestamp.
488            """
489         # First check for basic overlap using timestamp info given.
```

```
490             stream_id = self._stream_id(path)
491             iset = self._get_intervals(stream_id)
492             interval = Interval(start, end)
493             if iset.intersects(interval):
494                 raise OverlapError("new data overlaps existing data at range: "
495                                    + str(iset & interval))
496 
497             # Tenatively append the data.  This will raise a ValueError if
498             # there are any parse errors.
499             table = self.data.getnode(path)
500             row_start = table.nrows
501             table.append_data(data, start, end, binary)
502             row_end = table.nrows
503 
504             # Insert the record into the sql database.
505             self._add_interval(stream_id, interval, row_start, row_end)
506 
507             # And that's all
508             return
509 
510         def _find_start(self, table, dbinterval):
511             """
512             Given a DBInterval, find the row in the database that
513             corresponds to the start time.  Return the first database
514             position with a timestamp (first element) greater than or
515             equal to 'start'.
516             """
517             # Optimization for the common case where an interval wasn't truncated
518             if dbinterval.start == dbinterval.db_start:
519                 return dbinterval.db_startpos
520             return bisect.bisect_left(table,
521                                       dbinterval.start,
522                                       dbinterval.db_startpos,
523                                       dbinterval.db_endpos)
524 
525         def _find_end(self, table, dbinterval):
526             """
527             Given a DBInterval, find the row in the database that follows
528             the end time.  Return the first database position after the
529             row with timestamp (first element) greater than or equal
530             to 'end'.
531             """
532             # Optimization for the common case where an interval wasn't truncated
533             if dbinterval.end == dbinterval.db_end:
534                 return dbinterval.db_endpos
535             # Note that we still use bisect_left here, because we don't
536             # want to include the given timestamp in the results.  This is
537             # so a queries like 1:00 -> 2:00 and 2:00 -> 3:00 return
538             # non-overlapping data.
539             return bisect.bisect_left(table,
540                                       dbinterval.end,
541                                       dbinterval.db_startpos,
542                                       dbinterval.db_endpos)
543 
544         def stream_extract(self, path, start = None, end = None,
545                            count = False, markup = False, binary = False):
546             """
547             Returns (data, restart) tuple.
548 
549             'data' is ASCII-formatted data from the database, formatted
550             according to the layout of the stream.
551 
552             'restart', if not None, means that there were too many results to
553             return in a single request.  The data is complete from the
554             starting timestamp to the point at which it was truncated,
555             and a new request with a start time of 'restart' will fetch
556             the next block of data.
```

```
557
558             'count', if true, means to not return raw data, but just the count
559             of rows that would have been returned.  This is much faster
560             than actually fetching the data.  It is not limited by
561             max_results.
562
563             'markup', if true, indicates that returned data should be
564             marked with a comment denoting when a particular interval
565             starts, and another comment when an interval ends.
566
567             'binary', if true, means to return raw binary rather than
568             ASCII-formatted data.
569             """
570             stream_id = self._stream_id(path)
571             table = self.data.getnode(path)
572             intervals = self._get_intervals(stream_id)
573             (start, end) = self._check_user_times(start, end)
574             requested = Interval(start, end)
575             result = []
576             matched = 0
577             remaining = self.max_results
578             restart = None
579             if binary and (markup or count):
580                 raise NilmDBError("binary mode can't be used with markup or count")
581             for interval in intervals.intersection(requested):
582                 # Reading single rows from the table is too slow, so
583                 # we use two bisections to find both the starting and
584                 # ending row for this particular interval, then
585                 # read the entire range as one slice.
586                 row_start = self._find_start(table, interval)
587                 row_end = self._find_end(table, interval)
588
589                 if count:
590                     matched += row_end - row_start
591                     continue
592
593                 # Shorten it if we'll hit the maximum number of results
594                 row_max = row_start + remaining
595                 if row_max < row_end:
596                     row_end = row_max
597                     restart = table[row_max]
598
599                 # Add markup
600                 if markup:
601                     result.append("# interval-start " +
602                                   timestamp_to_string(interval.start) + "\n")
603
604                 # Gather these results up
605                 result.append(table.get_data(row_start, row_end, binary))
606
607                 # Count them
608                 remaining -= row_end - row_start
609
610                 # Add markup, and exit if restart is set.
611                 if restart is not None:
612                     if markup:
613                         result.append("# interval-end " +
614                                       timestamp_to_string(restart) + "\n")
615                     break
616                 if markup:
617                     result.append("# interval-end " +
618                                   timestamp_to_string(interval.end) + "\n")
619
620             if count:
621                 return matched
622             return ("".join(result), restart)
623
```

```
624     def stream_remove(self, path, start = None, end = None):
625         """
626         Remove data from the specified time interval within a stream.
627 
628         Removes data in the interval [start, end), and intervals are
629         truncated or split appropriately.
630 
631         Returns a (removed, restart) tuple.
632 
633         'removed' is the number of data points that were removed.
634 
635         'restart', if not None, means there were too many rows to
636         remove in a single request.  This function should be called
637         again with a start time of 'restart' to complete the removal.
638         """
639         stream_id = self._stream_id(path)
640         table = self.data.getnode(path)
641         intervals = self._get_intervals(stream_id)
642         (start, end) = self._check_user_times(start, end)
643         to_remove = Interval(start, end)
644         removed = 0
645         remaining = self.max_removals
646         restart = None
647 
648         # Can't remove intervals from within the iterator, so we need to
649         # remember what's currently in the intersection now.
650         all_candidates = list(intervals.intersection(to_remove, orig = True))
651 
652         for (dbint, orig) in all_candidates:
653             # Find row start and end
654             row_start = self._find_start(table, dbint)
655             row_end = self._find_end(table, dbint)
656 
657             # Shorten it if we'll hit the maximum number of removals
658             row_max = row_start + remaining
659             if row_max < row_end:
660                 row_end = row_max
661                 dbint.end = table[row_max]
662                 restart = dbint.end
663 
664             # Adjust the DBInterval to match the newly found ends
665             dbint.db_start = dbint.start
666             dbint.db_end = dbint.end
667             dbint.db_startpos = row_start
668             dbint.db_endpos = row_end
669 
670             # Remove interval from the database
671             self._remove_interval(stream_id, orig, dbint)
672 
673             # Remove data from the underlying table storage
674             table.remove(row_start, row_end)
675 
676             # Count how many were removed
677             removed += row_end - row_start
678             remaining -= row_end - row_start
679 
680             if restart is not None:
681                 break
682 
683         return (removed, restart)
```

Listing B-4: `nilmdb/server/serverutil.py`: HTTP server utilities. These are various decorators and utilities used by the HTTP server.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/server/serverutil.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
"""Miscellaneous decorators and other helpers for running a CherryPy
server"""

import cherrypy
import sys
import os
import decorator
import simplejson as json

# Helper to parse parameters into booleans
def bool_param(s):
    """Return a bool indicating whether parameter 's' was True or False,
    supporting a few different types for 's'."""
    try:
        ss = s.lower()
        if ss in [ "0", "false", "f", "no", "n" ]:
            return False
        if ss in [ "1", "true", "t", "yes", "y" ]:
            return True
    except Exception:
        return bool(s)
    raise cherrypy.HTTPError("400 Bad Request",
                             "can't parse parameter: " + ss)

# Decorators
def chunked_response(func):
    """Decorator to enable chunked responses."""
    # Set this to False to get better tracebacks from some requests
    # (/stream/extract, /stream/intervals).
    func._cp_config = { 'response.stream': True }
    return func

def response_type(content_type):
    """Return a decorator-generating function that sets the
    response type to the specified string."""
    def wrapper(func, *args, **kwargs):
        cherrypy.response.headers['Content-Type'] = content_type
        return func(*args, **kwargs)
    return decorator.decorator(wrapper)

@decorator.decorator
def workaround_cp_bug_1200(func, *args, **kwargs): # pragma: no cover
    """Decorator to work around CherryPy bug #1200 in a response
    generator.

    Even if chunked responses are disabled, LookupError or
    UnicodeError exceptions may still be swallowed by CherryPy due to
    bug #1200.  This throws them as generic Exceptions instead so that
    they make it through.
    """
    exc_info = None
    try:
        for val in func(*args, **kwargs):
            yield val
    except (LookupError, UnicodeError):
        # Re-raise it, but maintain the original traceback
        exc_info = sys.exc_info()
        new_exc = Exception(exc_info[0].__name__ + ": " + str(exc_info[1]))
        raise new_exc, None, exc_info[2]
```

```
    finally:
        del exc_info

def exception_to_httperror(*expected):
    """Return a decorator-generating function that catches expected
    errors and throws a HTTPError describing it instead.

        @exception_to_httperror(NilmDBError, ValueError)
        def foo():
            pass
    """
    def wrapper(func, *args, **kwargs):
        exc_info = None
        try:
            return func(*args, **kwargs)
        except expected:
            # Re-raise it, but maintain the original traceback
            exc_info = sys.exc_info()
            new_exc = cherrypy.HTTPError("400 Bad Request", str(exc_info[1]))
            raise new_exc, None, exc_info[2]
        finally:
            del exc_info
    # We need to preserve the function's argspecs for CherryPy to
    # handle argument errors correctly.  Decorator.decorator takes
    # care of that.
    return decorator.decorator(wrapper)

# Custom CherryPy tools

def CORS_allow(methods):
    """This does several things:

    Handles CORS preflight requests.
    Adds Allow: header to all requests.
    Raise 405 if request.method not in method.

    It is similar to cherrypy.tools.allow, with the CORS stuff added.

    Add this to CherryPy with:
    cherrypy.tools.CORS_allow = cherrypy.Tool('on_start_resource', CORS_allow)
    """
    request = cherrypy.request.headers
    response = cherrypy.response.headers

    if not isinstance(methods, (tuple, list)): # pragma: no cover
        methods = [ methods ]
    methods = [ m.upper() for m in methods if m ]
    if not methods: # pragma: no cover
        methods = [ 'GET', 'HEAD' ]
    elif 'GET' in methods and 'HEAD' not in methods: # pragma: no cover
        methods.append('HEAD')
    response['Allow'] = ', '.join(methods)

    # Allow all origins
    if 'Origin' in request:
        response['Access-Control-Allow-Origin'] = request['Origin']

    # If it's a CORS request, send response.
    request_method = request.get("Access-Control-Request-Method", None)
    request_headers = request.get("Access-Control-Request-Headers", None)
    if (cherrypy.request.method == "OPTIONS" and
        request_method and request_headers):
        response['Access-Control-Allow-Headers'] = request_headers
        response['Access-Control-Allow-Methods'] = ', '.join(methods)
        # Try to stop further processing and return a 200 OK
        cherrypy.response.status = "200 OK"
        cherrypy.response.body = ""
```

```
        cherrypy.request.handler = lambda: ""
        return

    # Reject methods that were not explicitly allowed
    if cherrypy.request.method not in methods:
        raise cherrypy.HTTPError(405)


# Helper for json_in tool to process JSON data into normal request
# parameters.
def json_to_request_params(body):
    cherrypy.lib.jsontools.json_processor(body)
    if not isinstance(cherrypy.request.json, dict):
        raise cherrypy.HTTPError(415)
    cherrypy.request.params.update(cherrypy.request.json)

# Used as an "error_page.default" handler
def json_error_page(status, message, traceback, version,
                    force_traceback = False):
    """Return a custom error page in JSON so the client can parse it"""
    errordata = { "status" : status,
                  "message" : message,
                  "traceback" : traceback }
    # Don't send a traceback if the error was 400-499 (client's fault)
    try:
        code = int(status.split()[0])
        if not force_traceback:
            if code >= 400 and code <= 499:
                errordata["traceback"] = ""
    except Exception: # pragma: no cover
        pass
    # Override the response type, which was previously set to text/html
    cherrypy.serving.response.headers['Content-Type'] = (
        "application/json;charset=utf-8" )
    # Undo the HTML escaping that cherrypy's get_error_page function applies
    # (cherrypy issue 1135)
    for k, v in errordata.iteritems():
        v = v.replace("&lt;","<")
        v = v.replace("&gt;",">")
        v = v.replace("&amp;","&")
        errordata[k] = v
    return json.dumps(errordata, separators=(',',':'))

# Start/stop CherryPy standalone server
def cherrypy_start(blocking = False, event = False, embedded = False):
    """Start the CherryPy server, handling errors and signals
    somewhat gracefully."""

    if not embedded: # pragma: no cover
        # Handle signals nicely
        if hasattr(cherrypy.engine, "signal_handler"):
            cherrypy.engine.signal_handler.subscribe()
        if hasattr(cherrypy.engine, "console_control_handler"):
            cherrypy.engine.console_control_handler.subscribe()

    # Cherrypy stupidly calls os._exit(70) when it can't bind the
    # port.  At least try to print a reasonable error and continue
    # in this case, rather than just dying silently (as we would
    # otherwise do in embedded mode)
    real_exit = os._exit
    def fake_exit(code): # pragma: no cover
        if code == os.EX_SOFTWARE:
            fprintf(sys.stderr, "error: CherryPy called os._exit!\n")
        else:
            real_exit(code)
    os._exit = fake_exit
    cherrypy.engine.start()
```

```
194     os._exit = real_exit
195
196     # Signal that the engine has started successfully
197     if event is not None:
198         event.set()
199
200     if blocking:
201         try:
202             cherrypy.engine.wait(cherrypy.engine.states.EXITING,
203                                  interval = 0.1, channel = 'main')
204         except (KeyboardInterrupt, IOError): # pragma: no cover
205             cherrypy.engine.log('Keyboard Interrupt: shutting down bus')
206             cherrypy.engine.exit()
207         except SystemExit: # pragma: no cover
208             cherrypy.engine.log('SystemExit raised: shutting down bus')
209             cherrypy.engine.exit()
210             raise
211
212 # Stop CherryPy server
213 def cherrypy_stop():
214     cherrypy.engine.exit()
```

Listing B-5: `nilmdb/server/server.py`: HTTP server interface and WSGI application server. This dispatches HTTP requests, and provides setup routines for both the standalone CherryPy server and the Apache WSGI application.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/server/server.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  """CherryPy-based server for accessing NILM database via HTTP"""
2
3  # Need absolute_import so that "import nilmdb" won't pull in
4  # nilmdb.py, but will pull the nilmdb module instead.
5  from __future__ import absolute_import
6  import nilmdb.server
7  from nilmdb.utils.printf import *
8  from nilmdb.server.errors import NilmDBError
9  from nilmdb.utils.time import string_to_timestamp
10
11 import cherrypy
12 import sys
13 import os
14 import socket
15 import simplejson as json
16 import decorator
17 import psutil
18 import traceback
19
20 from nilmdb.server.serverutil import (
21     chunked_response,
22     response_type,
23     workaround_cp_bug_1200,
24     exception_to_httperror,
25     CORS_allow,
26     json_to_request_params,
27     json_error_page,
28     cherrypy_start,
29     cherrypy_stop,
30     bool_param,
31     )
```

```
32
33  # Add CORS_allow tool
34  cherrypy.tools.CORS_allow = cherrypy.Tool('on_start_resource', CORS_allow)
35
36  class NilmApp(object):
37      def __init__(self, db):
38          self.db = db
39
40  # CherryPy apps
41  class Root(NilmApp):
42      """Root application for NILM database"""
43
44      def __init__(self, db):
45          super(Root, self).__init__(db)
46
47      # /
48      @cherrypy.expose
49      def index(self):
50          cherrypy.response.headers['Content-Type'] = 'text/plain'
51          msg = sprintf("This is NilmDB version %s, running on host %s.\n",
52                        nilmdb.__version__, socket.getfqdn())
53          return msg
54
55      # /favicon.ico
56      @cherrypy.expose
57      def favicon_ico(self):
58          raise cherrypy.NotFound()
59
60      # /version
61      @cherrypy.expose
62      @cherrypy.tools.json_out()
63      def version(self):
64          return nilmdb.__version__
65
66      # /dbinfo
67      @cherrypy.expose
68      @cherrypy.tools.json_out()
69      def dbinfo(self):
70          """Return a dictionary with the database path,
71          size of the database in bytes, and free disk space in bytes"""
72          path = self.db.get_basepath()
73          usage = psutil.disk_usage(path)
74          dbsize = nilmdb.utils.du(path)
75          return { "path": path,
76                   "size": dbsize,
77                   "other": max(usage.used - dbsize, 0),
78                   "reserved": max(usage.total - usage.used - usage.free, 0),
79                   "free": usage.free }
80
81  class Stream(NilmApp):
82      """Stream-specific operations"""
83
84      # Helpers
85      def _get_times(self, start_param, end_param):
86          (start, end) = (None, None)
87          try:
88              if start_param is not None:
89                  start = string_to_timestamp(start_param)
90          except Exception:
91              raise cherrypy.HTTPError("400 Bad Request", sprintf(
92                  "invalid start (%s): must be a numeric timestamp", start_param))
93          try:
94              if end_param is not None:
95                  end = string_to_timestamp(end_param)
96          except Exception:
97              raise cherrypy.HTTPError("400 Bad Request", sprintf(
98                  "invalid end (%s): must be a numeric timestamp", end_param))
```

```
99          if start is not None and end is not None:
100             if start >= end:
101                 raise cherrypy.HTTPError(
102                     "400 Bad Request",
103                     sprintf("start must precede end (%s >= %s)",
104                             start_param, end_param))
105         return (start, end)
106
107     # /stream/list
108     # /stream/list?layout=float32_8
109     # /stream/list?path=/newton/prep&extended=1
110     @cherrypy.expose
111     @cherrypy.tools.json_out()
112     def list(self, path = None, layout = None, extended = None):
113         """List all streams in the database.  With optional path or
114         layout parameter, just list streams that match the given path
115         or layout.
116
117         If extended is missing or zero, returns a list of lists
118         containing the path and layout: [ path, layout ]
119
120         If extended is true, returns a list of lists containing
121         extended info: [ path, layout, extent_min, extent_max,
122         total_rows, total_seconds ].  More data may be added.
123         """
124         return self.db.stream_list(path, layout, bool(extended))
125
126     # /stream/create?path=/newton/prep&layout=float32_8
127     @cherrypy.expose
128     @cherrypy.tools.json_in()
129     @cherrypy.tools.json_out()
130     @exception_to_httperror(NilmDBError, ValueError)
131     @cherrypy.tools.CORS_allow(methods = ["POST"])
132     def create(self, path, layout):
133         """Create a new stream in the database.  Provide path
134         and one of the nilmdb.layout.layouts keys.
135         """
136         return self.db.stream_create(path, layout)
137
138     # /stream/destroy?path=/newton/prep
139     @cherrypy.expose
140     @cherrypy.tools.json_in()
141     @cherrypy.tools.json_out()
142     @exception_to_httperror(NilmDBError)
143     @cherrypy.tools.CORS_allow(methods = ["POST"])
144     def destroy(self, path):
145         """Delete a stream.  Fails if any data is still present."""
146         return self.db.stream_destroy(path)
147
148     # /stream/rename?oldpath=/newton/prep&newpath=/newton/prep/1
149     @cherrypy.expose
150     @cherrypy.tools.json_in()
151     @cherrypy.tools.json_out()
152     @exception_to_httperror(NilmDBError, ValueError)
153     @cherrypy.tools.CORS_allow(methods = ["POST"])
154     def rename(self, oldpath, newpath):
155         """Rename a stream."""
156         return self.db.stream_rename(oldpath, newpath)
157
158     # /stream/get_metadata?path=/newton/prep
159     # /stream/get_metadata?path=/newton/prep&key=foo&key=bar
160     @cherrypy.expose
161     @cherrypy.tools.json_out()
162     def get_metadata(self, path, key=None):
163         """Get metadata for the named stream.  If optional
164         key parameters are specified, only return metadata
165         matching the given keys."""
```

```
166             try:
167                 data = self.db.stream_get_metadata(path)
168             except nilmdb.server.nilmdb.StreamError as e:
169                 raise cherrypy.HTTPError("404 Not Found", e.message)
170             if key is None:  # If no keys specified, return them all
171                 key = data.keys()
172             elif not isinstance(key, list):
173                 key = [ key ]
174             result = {}
175             for k in key:
176                 if k in data:
177                     result[k] = data[k]
178                 else: # Return "None" for keys with no matching value
179                     result[k] = None
180             return result
181 
182         # Helper for set_metadata and get_metadata
183         def _metadata_helper(self, function, path, data):
184             if not isinstance(data, dict):
185                 try:
186                     data = dict(json.loads(data))
187                 except TypeError as e:
188                     raise NilmDBError("can't parse 'data' parameter: " + e.message)
189             for key in data:
190                 if not (isinstance(data[key], basestring) or
191                         isinstance(data[key], float) or
192                         isinstance(data[key], int)):
193                     raise NilmDBError("metadata values must be a string or number")
194             function(path, data)
195 
196         # /stream/set_metadata?path=/newton/prep&data=<json>
197         @cherrypy.expose
198         @cherrypy.tools.json_in()
199         @cherrypy.tools.json_out()
200         @exception_to_httperror(NilmDBError, LookupError)
201         @cherrypy.tools.CORS_allow(methods = ["POST"])
202         def set_metadata(self, path, data):
203             """Set metadata for the named stream, replacing any existing
204             metadata.  Data can be json-encoded or a plain dictionary."""
205             self._metadata_helper(self.db.stream_set_metadata, path, data)
206 
207         # /stream/update_metadata?path=/newton/prep&data=<json>
208         @cherrypy.expose
209         @cherrypy.tools.json_in()
210         @cherrypy.tools.json_out()
211         @exception_to_httperror(NilmDBError, LookupError, ValueError)
212         @cherrypy.tools.CORS_allow(methods = ["POST"])
213         def update_metadata(self, path, data):
214             """Set metadata for the named stream, replacing any existing
215             metadata.  Data can be json-encoded or a plain dictionary."""
216             self._metadata_helper(self.db.stream_update_metadata, path, data)
217 
218         # /stream/insert?path=/newton/prep
219         @cherrypy.expose
220         @cherrypy.tools.json_out()
221         @exception_to_httperror(NilmDBError, ValueError)
222         @cherrypy.tools.CORS_allow(methods = ["PUT"])
223         def insert(self, path, start, end, binary = False):
224             """
225             Insert new data into the database.  Provide textual data
226             (matching the path's layout) as a HTTP PUT.
227 
228             If 'binary' is True, expect raw binary data, rather than lines
229             of ASCII-formatted data.  Raw binary data is always
230             little-endian and matches the database types (including an
231             int64 timestamp).
232             """
```

```
        binary = bool_param(binary)

        # Important that we always read the input before throwing any
        # errors, to keep lengths happy for persistent connections.
        # Note that CherryPy 3.2.2 has a bug where this fails for GET
        # requests, if we ever want to handle those (issue #1134)
        body = cherrypy.request.body.read()

        # Verify content type for binary data
        content_type = cherrypy.request.headers.get('content-type')
        if binary and content_type:
            if content_type != "application/octet-stream":
                raise cherrypy.HTTPError("400", "Content type must be "
                                         "application/octet-stream for "
                                         "binary data, not " + content_type)

        # Check path and get layout
        if len(self.db.stream_list(path = path)) != 1:
            raise cherrypy.HTTPError("404", "No such stream: " + path)

        # Check limits
        (start, end) = self._get_times(start, end)

        # Pass the data directly to nilmdb, which will parse it and
        # raise a ValueError if there are any problems.
        self.db.stream_insert(path, start, end, body, binary)

        # Done
        return

    # /stream/remove?path=/newton/prep
    # /stream/remove?path=/newton/prep&start=1234567890.0&end=1234567899.0
    @cherrypy.expose
    @cherrypy.tools.json_in()
    @cherrypy.tools.CORS_allow(methods = ["POST"])
    @chunked_response
    @response_type("application/x-json-stream")
    def remove(self, path, start = None, end = None):
        """
        Remove data from the backend database.  Removes all data in
        the interval [start, end).

        Returns the number of data points removed.  Since this is a potentially
        long-running operation, multiple numbers may be returned as the
        data gets removed from the backend database.  The total number of
        points removed is the sum of all of these numbers.
        """
        (start, end) = self._get_times(start, end)

        if len(self.db.stream_list(path = path)) != 1:
            raise cherrypy.HTTPError("404", "No such stream: " + path)

        @workaround_cp_bug_1200
        def content(start, end):
            # Note: disable chunked responses to see tracebacks from here.
            while True:
                (removed, restart) = self.db.stream_remove(path, start, end)
                yield json.dumps(removed) + "\r\n"
                if restart is None:
                    break
                start = restart
        return content(start, end)

    # /stream/intervals?path=/newton/prep
    # /stream/intervals?path=/newton/prep&start=1234567890.0&end=1234567899.0
    # /stream/intervals?path=/newton/prep&diffpath=/newton/prep2
    @cherrypy.expose
```

```
300         @chunked_response
301         @response_type("application/x-json-stream")
302         def intervals(self, path, start = None, end = None, diffpath = None):
303             """
304             Get intervals from backend database.  Streams the resulting
305             intervals as JSON strings separated by CR LF pairs.  This may
306             make multiple requests to the nilmdb backend to avoid causing
307             it to block for too long.
308 
309             Returns intervals between 'start' and 'end' belonging to
310             'path'.  If 'diff' is provided, the set-difference between
311             intervals in 'path' and intervals in 'diffpath' are
312             returned instead.
313 
314             Note that the response type is the non-standard
315             'application/x-json-stream' for lack of a better option.
316             """
317             (start, end) = self._get_times(start, end)
318 
319             if len(self.db.stream_list(path = path)) != 1:
320                 raise cherrypy.HTTPError("404", "No such stream: " + path)
321 
322             if diffpath and len(self.db.stream_list(path = diffpath)) != 1:
323                 raise cherrypy.HTTPError("404", "No such stream: " + diffpath)
324 
325             @workaround_cp_bug_1200
326             def content(start, end):
327                 # Note: disable chunked responses to see tracebacks from here.
328                 while True:
329                     (ints, restart) = self.db.stream_intervals(path, start, end,
330                                                                diffpath)
331                     response = ''.join([ json.dumps(i) + "\r\n" for i in ints ])
332                     yield response
333                     if restart is None:
334                         break
335                     start = restart
336             return content(start, end)
337 
338         # /stream/extract?path=/newton/prep&start=1234567890.0&end=1234567899.0
339         @cherrypy.expose
340         @chunked_response
341         def extract(self, path, start = None, end = None,
342                     count = False, markup = False, binary = False):
343             """
344             Extract data from backend database.  Streams the resulting
345             entries as ASCII text lines separated by newlines.  This may
346             make multiple requests to the nilmdb backend to avoid causing
347             it to block for too long.
348 
349             If 'count' is True, returns a count rather than actual data.
350 
351             If 'markup' is True, adds comments to the stream denoting each
352             interval's start and end timestamp.
353 
354             If 'binary' is True, return raw binary data, rather than lines
355             of ASCII-formatted data.  Raw binary data is always
356             little-endian and matches the database types (including an
357             int64 timestamp).
358             """
359             binary = bool_param(binary)
360             markup = bool_param(markup)
361             count = bool_param(count)
362 
363             (start, end) = self._get_times(start, end)
364 
365             # Check path and get layout
366             if len(self.db.stream_list(path = path)) != 1:
```

```
367             raise cherrypy.HTTPError("404", "No such stream: " + path)
368
369         if binary:
370             content_type = "application/octet-stream"
371             if markup or count:
372                 raise cherrypy.HTTPError("400", "can't mix binary and "
373                                          "markup or count modes")
374         else:
375             content_type = "text/plain"
376         cherrypy.response.headers['Content-Type'] = content_type
377
378         @workaround_cp_bug_1200
379         def content(start, end):
380             # Note: disable chunked responses to see tracebacks from here.
381             if count:
382                 matched = self.db.stream_extract(path, start, end,
383                                                  count = True)
384                 yield sprintf("%d\n", matched)
385                 return
386
387             while True:
388                 (data, restart) = self.db.stream_extract(
389                     path, start, end, count = False,
390                     markup = markup, binary = binary)
391                 yield data
392
393                 if restart is None:
394                     return
395                 start = restart
396         return content(start, end)
397
398 class Exiter(object):
399     """App that exits the server, for testing"""
400     @cherrypy.expose
401     def index(self):
402         cherrypy.response.headers['Content-Type'] = 'text/plain'
403         def content():
404             yield 'Exiting by request'
405             raise SystemExit
406         return content()
407     index._cp_config = { 'response.stream': True }
408
409 class Server(object):
410     def __init__(self, db, host = '127.0.0.1', port = 8080,
411                  stoppable = False,       # whether /exit URL exists
412                  embedded = True,         # hide diagnostics and output, etc
413                  fast_shutdown = False,   # don't wait for clients to disconn.
414                  force_traceback = False, # include traceback in all errors
415                  basepath = '',           # base URL path for cherrypy.tree
416                  ):
417         # Save server version, just for verification during tests
418         self.version = nilmdb.__version__
419
420         self.embedded = embedded
421         self.db = db
422         if not getattr(db, "_thread_safe", None):
423             raise KeyError("Database object " + str(db) + " doesn't claim "
424                            "to be thread safe.  You should pass "
425                            "nilmdb.utils.serializer_proxy(NilmDB)(args) "
426                            "rather than NilmDB(args).")
427
428         # Build up global server configuration
429         cherrypy.config.update({
430             'server.socket_host': host,
431             'server.socket_port': port,
432             'engine.autoreload_on': False,
433             'server.max_request_body_size': 8*1024*1024,
```

```
            })
        if self.embedded:
            cherrypy.config.update({ 'environment': 'embedded' })

        # Build up application specific configuration
        app_config = {}
        app_config.update({
            'error_page.default': self.json_error_page,
            })

        # Some default headers to just help identify that things are working
        app_config.update({ 'response.headers.X-Jim-Is-Awesome': 'yeah' })

        # Set up Cross-Origin Resource Sharing (CORS) handler so we
        # can correctly respond to browsers' CORS preflight requests.
        # This also limits verbs to GET and HEAD by default.
        app_config.update({ 'tools.CORS_allow.on': True,
                            'tools.CORS_allow.methods': ['GET', 'HEAD'] })

        # Configure the 'json_in' tool to also allow other content-types
        # (like x-www-form-urlencoded), and to treat JSON as a dict that
        # fills requests.param.
        app_config.update({ 'tools.json_in.force': False,
                            'tools.json_in.processor': json_to_request_params })

        # Send tracebacks in error responses.  They're hidden by the
        # error_page function for client errors (code 400-499).
        app_config.update({ 'request.show_tracebacks' : True })
        self.force_traceback = force_traceback

        # Patch CherryPy error handler to never pad out error messages.
        # This isn't necessary, but then again, neither is padding the
        # error messages.
        cherrypy._cperror._ie_friendly_error_sizes = {}

        # Build up the application and mount it
        root = Root(self.db)
        root.stream = Stream(self.db)
        if stoppable:
            root.exit = Exiter()
        cherrypy.tree.apps = {}
        cherrypy.tree.mount(root, basepath, config = { "/" : app_config })

        # Shutdowns normally wait for clients to disconnect.  To speed
        # up tests, set fast_shutdown = True
        if fast_shutdown:
            # Setting timeout to 0 triggers os._exit(70) at shutdown, grr...
            cherrypy.server.shutdown_timeout = 0.01
        else:
            cherrypy.server.shutdown_timeout = 5

        # Set up the WSGI application pointer for external programs
        self.wsgi_application = cherrypy.tree

    def json_error_page(self, status, message, traceback, version):
        """Return a custom error page in JSON so the client can parse it"""
        return json_error_page(status, message, traceback, version,
                               self.force_traceback)

    def start(self, blocking = False, event = None):
        cherrypy_start(blocking, event, self.embedded)

    def stop(self):
        cherrypy_stop()

# Use a single global nilmdb.server.NilmDB and nilmdb.server.Server
# instance since the database can only be opened once.  For this to
```

```
# work, the web server must use only a single process and single
# Python interpreter.  Multiple threads are OK.
_wsgi_server = None
def wsgi_application(dbpath, basepath): # pragma: no cover
    """Return a WSGI application object with a database at the
    specified path.

    'dbpath' is a filesystem location, e.g. /home/nilm/db

    'basepath' is the URL path of the application base, which
    is the same as the first argument to Apache's WSGIScriptAlias
    directive.
    """
    def application(environ, start_response):
        global _wsgi_server
        if _wsgi_server is None:
            # Try to start the server
            try:
                db = nilmdb.utils.serializer_proxy(nilmdb.server.NilmDB)(dbpath)
                _wsgi_server = nilmdb.server.Server(
                    db, embedded = True,
                    basepath = basepath.rstrip('/'))
            except Exception:
                # Build an error message on failure
                import pprint
                err = sprintf("Initializing database at path '%s' failed:\n\n",
                              dbpath)
                err += traceback.format_exc()
                try:
                    import pwd
                    import grp
                    err += sprintf("\nRunning as: uid=%d (%s), gid=%d (%s) "
                                   "on host %s, pid %d\n",
                                   os.getuid(), pwd.getpwuid(os.getuid())[0],
                                   os.getgid(), grp.getgrgid(os.getgid())[0],
                                   socket.gethostname(), os.getpid())
                except ImportError:
                    pass
                err += sprintf("\nEnvironment:\n%s\n", pprint.pformat(environ))
        if _wsgi_server is None:
            # Serve up the error with our own mini WSGI app.
            headers = [ ('Content-type', 'text/plain'),
                        ('Content-length', str(len(err))) ]
            start_response("500 Internal Server Error", headers)
            return [err]

        # Call the normal application
        return _wsgi_server.wsgi_application(environ, start_response)
    return application
```

Listing B-6: `nilmdb/server/errors.py`: Shared NilmDB exceptions.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/server/errors.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
"""Exceptions"""

class NilmDBError(Exception):
    """Base exception for NilmDB errors"""
    def __init__(self, message = "Unspecified error"):
        Exception.__init__(self, message)

class StreamError(NilmDBError):
```

```
9       pass
10
11  class OverlapError(NilmDBError):
12      pass
```

Listing B-7: `nilmdb/server/interval.pyx`: Non-overlapping interval and interval set management. The supported operations on sets of intervals include insertion, subsetting, intersections, unions, and limited forms of removal. It is optimized for the usage patterns of NilmDB. This code is ia Cython file that is compiled to C for performance reasons.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/server/interval.pyx`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1   """Interval, IntervalSet
2
3   The Interval implemented here is just like
4   nilmdb.utils.interval.Interval, except implemented in Cython for
5   speed.
6
7   Represents an interval of time, and a set of such intervals.
8
9   Intervals are half-open, ie. they include data points with timestamps
10  [start, end)
11  """
12
13  # First implementation kept a sorted list of intervals and used
14  # biesct() to optimize some operations, but this was too slow.
15
16  # Second version was based on the quicksect implementation from
17  # python-bx, modified slightly to handle floating point intervals.
18  # This didn't support deletion.
19
20  # Third version is more similar to the first version, using a rb-tree
21  # instead of a simple sorted list to maintain O(log n) operations.
22
23  # Fourth version is an optimized rb-tree that stores interval starts
24  # and ends directly in the tree, like bxinterval did.
25
26  from ..utils.time import min_timestamp as nilmdb_min_timestamp
27  from ..utils.time import max_timestamp as nilmdb_max_timestamp
28  from ..utils.time import timestamp_to_string
29  from ..utils.iterator import imerge
30  from ..utils.interval import IntervalError
31  import itertools
32
33  cimport rbtree
34  from libc.stdint cimport uint64_t, int64_t
35
36  ctypedef int64_t timestamp_t
37
38  cdef class Interval:
39      """Represents an interval of time."""
40
41      cdef public timestamp_t start, end
42
43      def __init__(self, timestamp_t start, timestamp_t end):
44          """
45          'start' and 'end' are arbitrary numbers that represent time
```

```
        """
        if start >= end:
            # Explicitly disallow zero-width intervals (since they're half-open)
            raise IntervalError("start %s must precede end %s" % (start, end))
        self.start = start
        self.end = end

    def __repr__(self):
        s = repr(self.start) + ", " + repr(self.end)
        return self.__class__.__name__ + "(" + s + ")"

    def __str__(self):
        return ("[" + timestamp_to_string(self.start) +
                " -> " + timestamp_to_string(self.end) + ")")

    def __cmp__(self, Interval other):
        """Compare two intervals.  If non-equal, order by start then end"""
        return cmp(self.start, other.start) or cmp(self.end, other.end)

    cpdef intersects(self, Interval other):
        """Return True if two Interval objects intersect"""
        if (self.end <= other.start or self.start >= other.end):
            return False
        return True

    cpdef subset(self, timestamp_t start, timestamp_t end):
        """Return a new Interval that is a subset of this one"""
        # A subclass that tracks additional data might override this.
        if start < self.start or end > self.end:
            raise IntervalError("not a subset")
        return Interval(start, end)

cdef class DBInterval(Interval):
    """
    Like Interval, but also tracks corresponding start/end times and
    positions within the database.  These are not currently modified
    when subsets are taken, but can be used later to help zero in on
    database positions.

    The actual 'start' and 'end' will always fall within the database
    start and end, e.g.:
        db_start = 100, db_startpos = 10000
        start = 123
        end = 150
        db_end = 200, db_endpos = 20000
    """

    cpdef public timestamp_t db_start, db_end
    cpdef public uint64_t db_startpos, db_endpos

    def __init__(self, start, end,
                 db_start, db_end,
                 db_startpos, db_endpos):
        """
        'db_start' and 'db_end' are arbitrary numbers that represent
        time.  They must be a strict superset of the time interval
        covered by 'start' and 'end'.  The 'db_startpos' and
        'db_endpos' are arbitrary database position indicators that
        correspond to those points.
        """
        Interval.__init__(self, start, end)
        self.db_start = db_start
        self.db_end = db_end
        self.db_startpos = db_startpos
        self.db_endpos = db_endpos
        if db_start > start or db_end < end:
            raise IntervalError("database times must span the interval times")
```

```
113
114     def __repr__(self):
115         s = repr(self.start) + ", " + repr(self.end)
116         s += ", " + repr(self.db_start) + ", " + repr(self.db_end)
117         s += ", " + repr(self.db_startpos) + ", " + repr(self.db_endpos)
118         return self.__class__.__name__ + "(" + s + ")"
119
120     cpdef subset(self, timestamp_t start, timestamp_t end):
121         """
122         Return a new DBInterval that is a subset of this one
123         """
124         if start < self.start or end > self.end:
125             raise IntervalError("not a subset")
126         return DBInterval(start, end,
127                           self.db_start, self.db_end,
128                           self.db_startpos, self.db_endpos)
129
130 cdef class IntervalSet:
131     """
132     A non-intersecting set of intervals.
133     """
134
135     cdef public rbtree.RBTree tree
136
137     def __init__(self, source=None):
138         """
139         'source' is an Interval or IntervalSet to add.
140         """
141         self.tree = rbtree.RBTree()
142         if source is not None:
143             self += source
144
145     def __iter__(self):
146         for node in self.tree:
147             if node.obj:
148                 yield node.obj
149
150     def __len__(self):
151         return sum(1 for x in self)
152
153     def __repr__(self):
154         descs = [ repr(x) for x in self ]
155         return self.__class__.__name__ + "([" + ", ".join(descs) + "])"
156
157     def __str__(self):
158         descs = [ str(x) for x in self ]
159         return  "[" + ", ".join(descs) + "]"
160
161     def __match__(self, other):
162         # This isn't particularly efficient, but it shouldn't get used in the
163         # general case.
164         """Test equality of two IntervalSets.
165
166         Treats adjacent Intervals as equivalent to one long interval,
167         so this function really tests whether the IntervalSets cover
168         the same spans of time."""
169         i = 0
170         j = 0
171         outside = True
172
173         def is_adjacent(a, b):
174             """Return True if two Intervals are adjacent (same end or start)"""
175             if a.end == b.start or b.end == a.start:
176                 return True
177             else:
178                 return False
179
```

```
        this = list(self)
        that = list(other)

        try:
            while True:
                if (outside):
                    # To match, we need to be finished both sets
                    if (i >= len(this) and j >= len(that)):
                        return True
                    # Or the starts need to match
                    if (this[i].start != that[j].start):
                        return False
                    outside = False
                else:
                    # We can move on if the two interval ends match
                    if (this[i].end == that[j].end):
                        i += 1
                        j += 1
                        outside = True
                    else:
                        # Whichever ends first needs to be adjacent to the next
                        if (this[i].end < that[j].end):
                            if (not is_adjacent(this[i],this[i+1])):
                                return False
                            i += 1
                        else:
                            if (not is_adjacent(that[j],that[j+1])):
                                return False
                            j += 1
        except IndexError:
            return False

    # Use __richcmp__ instead of __eq__, __ne__ for Cython.
    def __richcmp__(self, other, int op):
        if op == 2: # ==
            return self.__match__(other)
        elif op == 3: # !=
            return not self.__match__(other)
        return False
    #def __eq__(self, other):
    #    return self.__match__(other)
    #
    #def __ne__(self, other):
    #    return not self.__match__(other)

    def __iadd__(self, object other not None):
        """Inplace add -- modifies self

        This throws an exception if the regions being added intersect."""
        if isinstance(other, Interval):
            if self.intersects(other):
                raise IntervalError("Tried to add overlapping interval "
                                    "to this set")
            self.tree.insert(rbtree.RBNode(other.start, other.end, other))
        else:
            for x in other:
                self.__iadd__(x)
        return self

    def iadd_nocheck(self, Interval other not None):
        """Inplace add -- modifies self.
        'Optimized' version that doesn't check for intersection and
        only inserts the new interval into the tree."""
        self.tree.insert(rbtree.RBNode(other.start, other.end, other))

    def __isub__(self, Interval other not None):
        """Inplace subtract -- modifies self
```

```
247
248         Removes an interval from the set.  Must exist exactly
249         as provided -- cannot remove a subset of an existing interval."""
250         i = self.tree.find(other.start, other.end)
251         if i is None:
252             raise IntervalError("interval " + str(other) + " not in tree")
253         self.tree.delete(i)
254         return self
255
256     def __add__(self, other not None):
257         """Add -- returns a new object"""
258         new = IntervalSet(self)
259         new += IntervalSet(other)
260         return new
261
262     def __and__(self, other not None):
263         """
264         Compute a new IntervalSet from the intersection of this
265         IntervalSet with one other interval.
266
267         Output intervals are built as subsets of the intervals in the
268         first argument (self).
269         """
270         out = IntervalSet()
271         for i in self.intersection(other):
272             out.tree.insert(rbtree.RBNode(i.start, i.end, i))
273         return out
274
275     def intersection(self, Interval interval not None, orig = False):
276         """
277         Compute a sequence of intervals that correspond to the
278         intersection between `self` and the provided interval.
279         Returns a generator that yields each of these intervals
280         in turn.
281
282         Output intervals are built as subsets of the intervals in the
283         first argument (self).
284
285         If orig = True, also return the original interval that was
286         (potentially) subsetted to make the one that is being
287         returned.
288         """
289         if orig:
290             for n in self.tree.intersect(interval.start, interval.end):
291                 i = n.obj
292                 subset = i.subset(max(i.start, interval.start),
293                                   min(i.end, interval.end))
294                 yield (subset, i)
295         else:
296             for n in self.tree.intersect(interval.start, interval.end):
297                 i = n.obj
298                 subset = i.subset(max(i.start, interval.start),
299                                   min(i.end, interval.end))
300                 yield subset
301
302     cpdef intersects(self, Interval other):
303         """Return True if this IntervalSet intersects another interval"""
304         for n in self.tree.intersect(other.start, other.end):
305             if n.obj.intersects(other):
306                 return True
307         return False
308
309     def find_end(self, timestamp_t t):
310         """
311         Return an Interval from this tree that ends at time t, or
312         None if it doesn't exist.
313         """
```

```
314         n = self.tree.find_left_end(t)
315         if n and n.obj.end == t:
316             return n.obj
317         return None
```

Listing B-8: `nilmdb/server/rbtree.pyx`: Red-black interval tree implementation. This code supports non-overlapping intervals, and provides a number of features used by the `Interval` class to efficiently manage sets. This code is a Cython file that is compiled to C for performance reasons.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/server/rbtree.pyx`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  # cython: profile=False
2  # cython: cdivision=True
3
4  """
5  Jim Paris <jim@jtan.com>
6
7  Red-black tree, where keys are stored as start/end timestamps.
8  This is a basic interval tree that holds half-open intervals:
9    [start, end)
10 Intervals must not overlap.  Fixing that would involve making this
11 into an augmented interval tree as described in CLRS 14.3.
12
13 Code that assumes non-overlapping intervals is marked with the
14 string 'non-overlapping'.
15 """
16
17 import sys
18 cimport rbtree
19
20 cdef class RBNode:
21     """One node of the Red/Black tree, containing a key (start, end)
22     and value (obj)"""
23     def __init__(self, double start, double end, object obj = None):
24         self.obj = obj
25         self.start = start
26         self.end = end
27         self.red = False
28         self.left = None
29         self.right = None
30
31     def __str__(self):
32         if self.red:
33             color = "R"
34         else:
35             color = "B"
36         if self.start == sys.float_info.min:
37             return "[node nil]"
38         return ("[node ("
39                 + str(self.obj) + ") "
40                 + str(self.start) + " -> " + str(self.end) + " "
41                 + color + "]")
42
43 cdef class RBTree:
44     """Red/Black tree"""
45
46     # Init
47     def __init__(self):
```

```
        self.nil = RBNode(start = sys.float_info.min,
                          end = sys.float_info.min)
        self.nil.left = self.nil
        self.nil.right = self.nil
        self.nil.parent = self.nil

        self.root = RBNode(start = sys.float_info.max,
                           end = sys.float_info.max)
        self.root.left = self.nil
        self.root.right = self.nil
        self.root.parent = self.nil

    # We have a dummy root node to simplify operations, so from an
    # external point of view, its left child is the real root.
    cpdef getroot(self):
        return self.root.left

    # Rotations and basic operations
    cdef void __rotate_left(self, RBNode x):
        """Rotate left:
        #   x           y
        #  / \   -->   / \
        # z   y       x   w
        #    / \     / \
        #   v   w   z   v
        """
        cdef RBNode y = x.right
        x.right = y.left
        if y.left is not self.nil:
            y.left.parent = x
        y.parent = x.parent
        if x is x.parent.left:
            x.parent.left = y
        else:
            x.parent.right = y
        y.left = x
        x.parent = y

    cdef void __rotate_right(self, RBNode y):
        """Rotate right:
        #     y           x
        #    / \   -->   / \
        #   x   w       z   y
        #  / \             / \
        # z   v           v   w
        """
        cdef RBNode x = y.left
        y.left = x.right
        if x.right is not self.nil:
            x.right.parent = y
        x.parent = y.parent
        if y is y.parent.left:
            y.parent.left = x
        else:
            y.parent.right = x
        x.right = y
        y.parent = x

    cdef RBNode __successor(self, RBNode x):
        """Returns the successor of RBNode x"""
        cdef RBNode y = x.right
        if y is not self.nil:
            while y.left is not self.nil:
                y = y.left
        else:
            y = x.parent
            while x is y.right:
```

```
                x = y
                y = y.parent
            if y is self.root:
                return self.nil
        return y
    cpdef RBNode successor(self, RBNode x):
        """Returns the successor of RBNode x, or None"""
        cdef RBNode y = self.__successor(x)
        return y if y is not self.nil else None

    cdef RBNode __predecessor(self, RBNode x):
        """Returns the predecessor of RBNode x"""
        cdef RBNode y = x.left
        if y is not self.nil:
            while y.right is not self.nil:
                y = y.right
        else:
            y = x.parent
            while x is y.left:
                if y is self.root:
                    y = self.nil
                    break
                x = y
                y = y.parent
        return y
    cpdef RBNode predecessor(self, RBNode x):
        """Returns the predecessor of RBNode x, or None"""
        cdef RBNode y = self.__predecessor(x)
        return y if y is not self.nil else None

    # Insertion
    cpdef insert(self, RBNode z):
        """Insert RBNode z into RBTree and rebalance as necessary"""
        z.left = self.nil
        z.right = self.nil
        cdef RBNode y = self.root
        cdef RBNode x = self.root.left
        while x is not self.nil:
            y = x
            if (x.start > z.start or (x.start == z.start and x.end > z.end)):
                x = x.left
            else:
                x = x.right
        z.parent = y
        if (y is self.root or
            (y.start > z.start or (y.start == z.start and y.end > z.end))):
            y.left = z
        else:
            y.right = z
        # relabel/rebalance
        self.__insert_fixup(z)

    cdef void __insert_fixup(self, RBNode x):
        """Rebalance/fix RBTree after a simple insertion of RBNode x"""
        x.red = True
        while x.parent.red:
            if x.parent is x.parent.parent.left:
                y = x.parent.parent.right
                if y.red:
                    x.parent.red = False
                    y.red = False
                    x.parent.parent.red = True
                    x = x.parent.parent
                else:
                    if x is x.parent.right:
                        x = x.parent
                        self.__rotate_left(x)
```

```
                    x.parent.red = False
                    x.parent.parent.red = True
                    self.__rotate_right(x.parent.parent)
            else: # same as above, left/right switched
                y = x.parent.parent.left
                if y.red:
                    x.parent.red = False
                    y.red = False
                    x.parent.parent.red = True
                    x = x.parent.parent
                else:
                    if x is x.parent.left:
                        x = x.parent
                        self.__rotate_right(x)
                    x.parent.red = False
                    x.parent.parent.red = True
                    self.__rotate_left(x.parent.parent)
        self.root.left.red = False

    # Deletion
    cpdef delete(self, RBNode z):
        if z.left is None or z.right is None:
            raise AttributeError("you can only delete a node object "
                                 + "from the tree; use find() to get one")
        cdef RBNode x, y
        if z.left is self.nil or z.right is self.nil:
            y = z
        else:
            y = self.__successor(z)
        if y.left is self.nil:
            x = y.right
        else:
            x = y.left
        x.parent = y.parent
        if x.parent is self.root:
            self.root.left = x
        else:
            if y is y.parent.left:
                y.parent.left = x
            else:
                y.parent.right = x
        if y is not z:
            # y is the node to splice out, x is its child
            y.left = z.left
            y.right = z.right
            y.parent = z.parent
            z.left.parent = y
            z.right.parent = y
            if z is z.parent.left:
                z.parent.left = y
            else:
                z.parent.right = y
            if not y.red:
                y.red = z.red
                self.__delete_fixup(x)
            else:
                y.red = z.red
        else:
            if not y.red:
                self.__delete_fixup(x)

    cdef void __delete_fixup(self, RBNode x):
        """Rebalance/fix RBTree after a deletion.  RBNode x is the
        child of the spliced out node."""
        cdef RBNode rootLeft = self.root.left
        while not x.red and x is not rootLeft:
            if x is x.parent.left:
```

```
                w = x.parent.right
                if w.red:
                    w.red = False
                    x.parent.red = True
                    self.__rotate_left(x.parent)
                    w = x.parent.right
                if not w.right.red and not w.left.red:
                    w.red = True
                    x = x.parent
                else:
                    if not w.right.red:
                        w.left.red = False
                        w.red = True
                        self.__rotate_right(w)
                        w = x.parent.right
                    w.red = x.parent.red
                    x.parent.red = False
                    w.right.red = False
                    self.__rotate_left(x.parent)
                    x = rootLeft # exit loop
            else: # same as above, left/right switched
                w = x.parent.left
                if w.red:
                    w.red = False
                    x.parent.red = True
                    self.__rotate_right(x.parent)
                    w = x.parent.left
                if not w.left.red and not w.right.red:
                    w.red = True
                    x = x.parent
                else:
                    if not w.left.red:
                        w.right.red = False
                        w.red = True
                        self.__rotate_left(w)
                        w = x.parent.left
                    w.red = x.parent.red
                    x.parent.red = False
                    w.left.red = False
                    self.__rotate_right(x.parent)
                    x = rootLeft # exit loop
        x.red = False

    # Walking, searching
    def __iter__(self):
        return self.inorder()

    def inorder(self, RBNode x = None):
        """Generator that performs an inorder walk for the tree
        rooted at RBNode x"""
        if x is None:
            x = self.getroot()
        while x.left is not self.nil:
            x = x.left
        while x is not self.nil:
            yield x
            x = self.__successor(x)

    cpdef RBNode find(self, double start, double end):
        """Return the node with exactly the given start and end."""
        cdef RBNode x = self.getroot()
        while x is not self.nil:
            if start < x.start:
                x = x.left
            elif start == x.start:
                if end == x.end:
                    break # found it
```

```
                elif end < x.end:
                    x = x.left
                else:
                    x = x.right
            else:
                x = x.right
        return x if x is not self.nil else None

    cpdef RBNode find_left_end(self, double t):
        """Find the leftmode node with end >= t.  With non-overlapping
        intervals, this is the first node that might overlap time t.

        Note that this relies on non-overlapping intervals, since
        it assumes that we can use the endpoints to traverse the
        tree even though it was created using the start points."""
        cdef RBNode x = self.getroot()
        while x is not self.nil:
            if t < x.end:
                if x.left is self.nil:
                    break
                x = x.left
            elif t == x.end:
                break
            else:
                if x.right is self.nil:
                    x = self.__successor(x)
                    break
                x = x.right
        return x if x is not self.nil else None

    cpdef RBNode find_right_start(self, double t):
        """Find the rightmode node with start <= t.  With non-overlapping
        intervals, this is the last node that might overlap time t."""
        cdef RBNode x = self.getroot()
        while x is not self.nil:
            if t < x.start:
                if x.left is self.nil:
                    x = self.__predecessor(x)
                    break
                x = x.left
            elif t == x.start:
                break
            else:
                if x.right is self.nil:
                    break
                x = x.right
        return x if x is not self.nil else None

    # Intersections
    def intersect(self, double start, double end):
        """Generator that returns nodes that overlap the given
        (start,end) range.  Assumes non-overlapping intervals."""
        # Start with the leftmode node that ends after start
        cdef RBNode n = self.find_left_end(start)
        while n is not None:
            if n.start >= end:
                # this node starts after the requested end; we're done
                break
            if start < n.end:
                # this node overlaps our requested area
                yield n
            n = self.successor(n)
```

Listing B-9: `nilmdb/server/bulkdata.py`: Bulk data storage interface. This code, along with the Rocket interface in Listing B-10, manages the lowest level of on-disk data storage, retrieval, and removal.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/server/bulkdata.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# Fixed record size bulk data storage

# Need absolute_import so that "import nilmdb" won't pull in
# nilmdb.py, but will pull the parent nilmdb module instead.
from __future__ import absolute_import
from __future__ import division
from nilmdb.utils.printf import *
from nilmdb.utils.time import timestamp_to_string as timestamp_to_string
import nilmdb.utils

import os
import cPickle as pickle
import re
import sys
import tempfile

import nilmdb.utils.lock
from . import rocket

# Up to 256 open file descriptors at any given time.
# These variables are global so they can be used in the decorator arguments.
table_cache_size = 32
fd_cache_size = 8

@nilmdb.utils.must_close(wrap_verify = False)
class BulkData(object):
    def __init__(self, basepath, **kwargs):
        self.basepath = basepath
        self.root = os.path.join(self.basepath, "data")
        self.lock = self.root + ".lock"
        self.lockfile = None

        # Tuneables
        if "file_size" in kwargs:
            self.file_size = kwargs["file_size"]
        else:
            # Default to approximately 128 MiB per file
            self.file_size = 128 * 1024 * 1024

        if "files_per_dir" in kwargs:
            self.files_per_dir = kwargs["files_per_dir"]
        else:
            # 32768 files per dir should work even on FAT32
            self.files_per_dir = 32768

        # Make root path
        if not os.path.isdir(self.root):
            os.mkdir(self.root)

        # Create the lock
        self.lockfile = open(self.lock, "w")
        if not nilmdb.utils.lock.exclusive_lock(self.lockfile):
            raise IOError('database at "' + self.basepath +
                          '" is already locked by another process')

    def close(self):
        self.getnode.cache_remove_all()
```

```
58          if self.lockfile:
59              nilmdb.utils.lock.exclusive_unlock(self.lockfile)
60              self.lockfile.close()
61              try:
62                  os.unlink(self.lock)
63              except OSError: # pragma: no cover
64                  pass
65              self.lockfile = None
66
67      def _encode_filename(self, path):
68          # Encode all paths to UTF-8, regardless of sys.getfilesystemencoding(),
69          # because we want to be able to represent all code points and the user
70          # will never be directly exposed to filenames.  We can then do path
71          # manipulations on the UTF-8 directly.
72          if isinstance(path, unicode):
73              return path.encode('utf-8')
74          return path
75
76      def _create_check_ospath(self, ospath):
77          if ospath[-1] == '/':
78              raise ValueError("invalid path; should not end with a /")
79          if Table.exists(ospath):
80              raise ValueError("stream already exists at this path")
81          if os.path.isdir(ospath):
82              # Look for any files in subdirectories.  Fully empty subdirectories
83              # are OK; they might be there during a rename
84              for (root, dirs, files) in os.walk(ospath):
85                  if len(files):
86                      raise ValueError(
87                          "non-empty subdirs of this path already exist")
88
89      def _create_parents(self, unicodepath):
90          """Verify the path name, and create parent directories if they
91          don't exist.  Returns a list of elements that got created."""
92          path = self._encode_filename(unicodepath)
93
94          if path[0] != '/':
95              raise ValueError("paths must start with /")
96          [ group, node ] = path.rsplit("/", 1)
97          if group == '':
98              raise ValueError("invalid path; path must contain at least one "
99                               "folder")
100         if node == '':
101             raise ValueError("invalid path; should not end with a /")
102         if not Table.valid_path(path):
103             raise ValueError("path name is invalid or contains reserved words")
104
105         # Create the table's base dir.  Note that we make a
106         # distinction here between NilmDB paths (always Unix style,
107         # split apart manually) and OS paths (built up with
108         # os.path.join)
109
110         # Make directories leading up to this one
111         elements = path.lstrip('/').split('/')
112         made_dirs = []
113         try:
114             # Make parent elements
115             for i in range(len(elements)):
116                 ospath = os.path.join(self.root, *elements[0:i])
117                 if Table.exists(ospath):
118                     raise ValueError("path is subdir of existing node")
119                 if not os.path.isdir(ospath):
120                     os.mkdir(ospath)
121                     made_dirs.append(ospath)
122         except Exception as e:
123             # Try to remove paths that we created; ignore errors
124             exc_info = sys.exc_info()
```

```
            for ospath in reversed(made_dirs): # pragma: no cover (hard to hit)
                try:
                    os.rmdir(ospath)
                except OSError:
                    pass
            raise exc_info[1], None, exc_info[2]

        return elements

    def create(self, unicodepath, layout_name):
        """
        unicodepath: path to the data (e.g. u'/newton/prep').
        Paths must contain at least two elements, e.g.:
           /newton/prep
           /newton/raw
           /newton/upstairs/prep
           /newton/upstairs/raw

        layout_name: string for nilmdb.layout.get_named(), e.g. 'float32_8'
        """
        elements = self._create_parents(unicodepath)

        # Make the final dir
        ospath = os.path.join(self.root, *elements)
        self._create_check_ospath(ospath)
        os.mkdir(ospath)

        try:
            # Write format string to file
            Table.create(ospath, layout_name, self.file_size,
                         self.files_per_dir)

            # Open and cache it
            self.getnode(unicodepath)
        except Exception:
            exc_info = sys.exc_info()
            try:
                os.rmdir(ospath)
            except OSError:
                pass
            raise exc_info[1], None, exc_info[2]

        # Success
        return

    def _remove_leaves(self, unicodepath):
        """Remove empty directories starting at the leaves of unicodepath"""
        path = self._encode_filename(unicodepath)
        elements = path.lstrip('/').split('/')
        for i in reversed(range(len(elements))):
            ospath = os.path.join(self.root, *elements[0:i+1])
            try:
                os.rmdir(ospath)
            except OSError:
                pass

    def rename(self, oldunicodepath, newunicodepath):
        """Move entire tree from 'oldunicodepath' to
        'newunicodepath'"""
        oldpath = self._encode_filename(oldunicodepath)
        newpath = self._encode_filename(newunicodepath)

        # Get OS paths
        oldelements = oldpath.lstrip('/').split('/')
        oldospath = os.path.join(self.root, *oldelements)
        newelements = newpath.lstrip('/').split('/')
        newospath = os.path.join(self.root, *newelements)
```

```
192
193         # Basic checks
194         if oldospath == newospath:
195             raise ValueError("old and new paths are the same")
196
197         # Remove Table object at old path from cache
198         self.getnode.cache_remove(self, oldunicodepath)
199
200         # Move the table to a temporary location
201         tmpdir = tempfile.mkdtemp(prefix = "rename-", dir = self.root)
202         tmppath = os.path.join(tmpdir, "table")
203         os.rename(oldospath, tmppath)
204
205         try:
206             # Check destination path
207             self._create_check_ospath(newospath)
208
209             # Create parent dirs for new location
210             self._create_parents(newunicodepath)
211
212             # Move table into new location
213             os.rename(tmppath, newospath)
214         except Exception:
215             # On failure, move the table back to original path
216             os.rename(tmppath, oldospath)
217             os.rmdir(tmpdir)
218             raise
219
220         # Prune old dirs
221         self._remove_leaves(oldunicodepath)
222         os.rmdir(tmpdir)
223
224     def destroy(self, unicodepath):
225         """Fully remove all data at a particular path.  No way to undo
226         it!  The group/path structure is removed, too."""
227         path = self._encode_filename(unicodepath)
228
229         # Get OS path
230         elements = path.lstrip('/').split('/')
231         ospath = os.path.join(self.root, *elements)
232
233         # Remove Table object from cache
234         self.getnode.cache_remove(self, unicodepath)
235
236         # Remove the contents of the target directory
237         if not Table.exists(ospath):
238             raise ValueError("nothing at that path")
239         for (root, dirs, files) in os.walk(ospath, topdown = False):
240             for name in files:
241                 os.remove(os.path.join(root, name))
242             for name in dirs:
243                 os.rmdir(os.path.join(root, name))
244
245         # Remove leftover empty directories
246         self._remove_leaves(unicodepath)
247
248     # Cache open tables
249     @nilmdb.utils.lru_cache(size = table_cache_size,
250                             onremove = lambda x: x.close())
251     def getnode(self, unicodepath):
252         """Return a Table object corresponding to the given database
253         path, which must exist."""
254         path = self._encode_filename(unicodepath)
255         elements = path.lstrip('/').split('/')
256         ospath = os.path.join(self.root, *elements)
257         return Table(ospath)
258
```

```
@nilmdb.utils.must_close(wrap_verify = False)
class Table(object):
    """Tools to help access a single table (data at a specific OS path)."""
    # See design.md for design details

    # Class methods, to help keep format details in this class.
    @classmethod
    def valid_path(cls, root):
        """Return True if a root path is a valid name"""
        return "_format" not in root.split("/")

    @classmethod
    def exists(cls, root):
        """Return True if a table appears to exist at this OS path"""
        return os.path.isfile(os.path.join(root, "_format"))

    @classmethod
    def create(cls, root, layout, file_size, files_per_dir):
        """Initialize a table at the given OS path with the
        given layout string"""

        # Calculate rows per file so that each file is approximately
        # file_size bytes.
        rkt = rocket.Rocket(layout, None)
        rows_per_file = max(file_size // rkt.binary_size, 1)
        rkt.close()

        fmt = { "rows_per_file": rows_per_file,
                "files_per_dir": files_per_dir,
                "layout": layout,
                "version": 3 }
        with open(os.path.join(root, "_format"), "wb") as f:
            pickle.dump(fmt, f, 2)

    # Normal methods
    def __init__(self, root):
        """'root' is the full OS path to the directory of this table"""
        self.root = root

        # Load the format
        with open(os.path.join(self.root, "_format"), "rb") as f:
            fmt = pickle.load(f)

        if fmt["version"] != 3: # pragma: no cover
            # Old versions used floating point timestamps, which aren't
            # valid anymore.
            raise NotImplementedError("old version " + str(fmt["version"]) +
                                      " bulk data store is not supported")

        self.rows_per_file = fmt["rows_per_file"]
        self.files_per_dir = fmt["files_per_dir"]
        self.layout = fmt["layout"]

        # Use rocket to get row size and file size
        rkt = rocket.Rocket(self.layout, None)
        self.row_size = rkt.binary_size
        self.file_size = rkt.binary_size * self.rows_per_file
        rkt.close()

        # Find nrows
        self.nrows = self._get_nrows()

    def close(self):
        self.file_open.cache_remove_all()

    # Internal helpers
    def _get_nrows(self):
```

```
        """Find nrows by locating the lexicographically last filename
        and using its size"""
        # Note that this just finds a 'nrows' that is guaranteed to be
        # greater than the row number of any piece of data that
        # currently exists, not necessarily all data that _ever_
        # existed.
        regex = re.compile("^[0-9a-f]{4,}$")

        # Find the last directory.  We sort and loop through all of them,
        # starting with the numerically greatest, because the dirs could be
        # empty if something was deleted but the directory was unexpectedly
        # not deleted.
        subdirs = sorted(filter(regex.search, os.listdir(self.root)),
                         key = lambda x: int(x, 16), reverse = True)

        for subdir in subdirs:
            # Now find the last file in that dir
            path = os.path.join(self.root, subdir)
            files = filter(regex.search, os.listdir(path))
            if not files: # pragma: no cover (shouldn't occur)
                # Empty dir: try the next one
                continue

            # Find the numerical max
            filename = max(files, key = lambda x: int(x, 16))
            offset = os.path.getsize(os.path.join(self.root, subdir, filename))

            # Convert to row number
            return self._row_from_offset(subdir, filename, offset)

        # No files, so no data
        return 0

    def _offset_from_row(self, row):
        """Return a (subdir, filename, offset, count) tuple:

          subdir: subdirectory for the file
        filename: the filename that contains the specified row
          offset: byte offset of the specified row within the file
           count: number of rows (starting at offset) that fit in the file
        """
        filenum = row // self.rows_per_file
        # It's OK if these format specifiers are too short; the filenames
        # will just get longer but will still sort correctly.
        dirname = sprintf("%04x", filenum // self.files_per_dir)
        filename = sprintf("%04x", filenum % self.files_per_dir)
        offset = (row % self.rows_per_file) * self.row_size
        count = self.rows_per_file - (row % self.rows_per_file)
        return (dirname, filename, offset, count)

    def _row_from_offset(self, subdir, filename, offset):
        """Return the row number that corresponds to the given
        'subdir/filename' and byte-offset within that file."""
        if (offset % self.row_size) != 0: # pragma: no cover
            # this shouldn't occur, unless there is some corruption somewhere
            raise ValueError("file offset is not a multiple of data size")
        filenum = int(subdir, 16) * self.files_per_dir + int(filename, 16)
        row = (filenum * self.rows_per_file) + (offset // self.row_size)
        return row

    def _remove_or_truncate_file(self, subdir, filename, offset = 0):
        """Remove the given file, and remove the subdirectory too
        if it's empty.  If offset is nonzero, truncate the file
        to that size instead."""
        # Close potentially open file in file_open LRU cache
        self.file_open.cache_remove(self, subdir, filename)
        if offset:
```

```
393             # Truncate it
394             with open(os.path.join(self.root, subdir, filename), "r+b") as f:
395                 f.truncate(offset)
396         else:
397             # Remove file
398             os.remove(os.path.join(self.root, subdir, filename))
399             # Try deleting subdir, too
400             try:
401                 os.rmdir(os.path.join(self.root, subdir))
402             except Exception:
403                 pass
404 
405     # Cache open files
406     @nilmdb.utils.lru_cache(size = fd_cache_size,
407                             onremove = lambda f: f.close())
408     def file_open(self, subdir, filename):
409         """Open and map a given 'subdir/filename' (relative to self.root).
410         Will be automatically closed when evicted from the cache."""
411         # Create path if it doesn't exist
412         try:
413             os.mkdir(os.path.join(self.root, subdir))
414         except OSError:
415             pass
416         # Return a rocket.Rocket object, which contains the open file
417         return rocket.Rocket(self.layout,
418                              os.path.join(self.root, subdir, filename))
419 
420     def append_data(self, data, start, end, binary = False):
421         """Parse the formatted string in 'data', according to the
422         current layout, and append it to the table.  If any timestamps
423         are non-monotonic, or don't fall between 'start' and 'end',
424         a ValueError is raised.
425 
426         If 'binary' is True, the data should be in raw binary format
427         instead: little-endian, matching the current table's layout,
428         including the int64 timestamp.
429 
430         If this function succeeds, it returns normally.  Otherwise,
431         the table is reverted back to its original state by truncating
432         or deleting files as necessary."""
433         data_offset = 0
434         last_timestamp = nilmdb.utils.time.min_timestamp
435         tot_rows = self.nrows
436         count = 0
437         linenum = 0
438         try:
439             while data_offset < len(data):
440                 # See how many rows we can fit into the current file,
441                 # and open it
442                 (subdir, fname, offset, count) = self._offset_from_row(tot_rows)
443                 f = self.file_open(subdir, fname)
444 
445                 # Ask the rocket object to parse and append up to "count"
446                 # rows of data, verifying things along the way.
447                 try:
448                     if binary:
449                         appender = f.append_binary
450                     else:
451                         appender = f.append_string
452                     (added_rows, data_offset, last_timestamp, linenum
453                      ) = appender(count, data, data_offset, linenum,
454                                   start, end, last_timestamp)
455                 except rocket.ParseError as e:
456                     (linenum, colnum, errtype, obj) = e.args
457                     if binary:
458                         where = "byte %d: " % (linenum)
459                     else:
```

```
                        where = "line %d, column %d: " % (linenum, colnum)
                    # Extract out the error line, add column marker
                    try:
                        if binary:
                            raise IndexError
                        bad = data.splitlines()[linenum-1]
                        bad += '\n' + ' ' * (colnum - 1) + '^'
                    except IndexError:
                        bad = ""
                    if errtype == rocket.ERR_NON_MONOTONIC:
                        err = "timestamp is not monotonically increasing"
                    elif errtype == rocket.ERR_OUT_OF_INTERVAL:
                        if obj < start:
                            err = sprintf("Data timestamp %s < start time %s",
                                          timestamp_to_string(obj),
                                          timestamp_to_string(start))
                        else:
                            err = sprintf("Data timestamp %s >= end time %s",
                                          timestamp_to_string(obj),
                                          timestamp_to_string(end))
                    else:
                        err = str(obj)
                    raise ValueError("error parsing input data: " +
                                     where + err + "\n" + bad)
                tot_rows += added_rows
        except Exception:
            # Some failure, so try to roll things back by truncating or
            # deleting files that we may have appended data to.
            cleanpos = self.nrows
            while cleanpos <= tot_rows:
                (subdir, fname, offset, count) = self._offset_from_row(cleanpos)
                self._remove_or_truncate_file(subdir, fname, offset)
                cleanpos += count
            # Re-raise original exception
            raise
        else:
            # Success, so update self.nrows accordingly
            self.nrows = tot_rows

    def get_data(self, start, stop, binary = False):
        """Extract data corresponding to Python range [n:m],
        and returns a formatted string"""
        if (start is None or
            stop is None or
            start > stop or
            start < 0 or
            stop > self.nrows):
            raise IndexError("Index out of range")

        ret = []
        row = start
        remaining = stop - start
        while remaining > 0:
            (subdir, filename, offset, count) = self._offset_from_row(row)
            if count > remaining:
                count = remaining
            f = self.file_open(subdir, filename)
            if binary:
                ret.append(f.extract_binary(offset, count))
            else:
                ret.append(f.extract_string(offset, count))
            remaining -= count
            row += count
        return b"".join(ret)

    def __getitem__(self, row):
        """Extract timestamps from a row, with table[n] notation."""
```

```
        if row < 0 or row >= self.nrows:
            raise IndexError("Index out of range")
        (subdir, filename, offset, count) = self._offset_from_row(row)
        f = self.file_open(subdir, filename)
        return f.extract_timestamp(offset)

    def _remove_rows(self, subdir, filename, start, stop):
        """Helper to mark specific rows as being removed from a
        file, and potentially remove or truncate the file itself."""
        # Close potentially open file in file_open LRU cache
        self.file_open.cache_remove(self, subdir, filename)

        # We keep a file like 0000.removed that contains a list of
        # which rows have been "removed".  Note that we never have to
        # remove entries from this list, because we never decrease
        # self.nrows, and so we will never overwrite those locations in the
        # file.  Only when the list covers the entire extent of the
        # file will that file be removed.
        datafile = os.path.join(self.root, subdir, filename)
        cachefile = datafile + ".removed"
        try:
            with open(cachefile, "rb") as f:
                ranges = pickle.load(f)
            cachefile_present = True
        except Exception:
            ranges = []
            cachefile_present = False

        # Append our new range and sort
        ranges.append((start, stop))
        ranges.sort()

        # Merge adjacent ranges into "out"
        merged = []
        prev = None
        for new in ranges:
            if prev is None:
                # No previous range, so remember this one
                prev = new
            elif prev[1] == new[0]:
                # Previous range connected to this new one; extend prev
                prev = (prev[0], new[1])
            else:
                # Not connected; append previous and start again
                merged.append(prev)
                prev = new
        if prev is not None:
            merged.append(prev)

        # If the range covered the whole file, we can delete it now.
        # Note that the last file in a table may be only partially
        # full (smaller than self.rows_per_file).  We purposely leave
        # those files around rather than deleting them, because the
        # remainder will be filled on a subsequent append(), and things
        # are generally easier if we don't have to special-case that.
        if (len(merged) == 1 and
            merged[0][0] == 0 and merged[0][1] == self.rows_per_file):
            # Delete files
            if cachefile_present:
                os.remove(cachefile)
            self._remove_or_truncate_file(subdir, filename, 0)
        else:
            # File needs to stick around.  This means we can get
            # degenerate cases where we have large files containing as
            # little as one row.  Try to punch a hole in the file,
            # so that this region doesn't take up filesystem space.
            offset = start * self.row_size
```

```
594             count = (stop - start) * self.row_size
595             nilmdb.utils.fallocate.punch_hole(datafile, offset, count)
596
597             # Update cache.  Try to do it atomically.
598             nilmdb.utils.atomic.replace_file(cachefile,
599                                              pickle.dumps(merged, 2))
600
601     def remove(self, start, stop):
602         """Remove specified rows [start, stop) from this table.
603
604         If a file is left empty, it is fully removed.  Otherwise, a
605         parallel data file is used to remember which rows have been
606         removed, and the file is otherwise untouched."""
607         if start < 0 or start > stop or stop > self.nrows:
608             raise IndexError("Index out of range")
609
610         row = start
611         remaining = stop - start
612         while remaining:
613             # Loop through each file that we need to touch
614             (subdir, filename, offset, count) = self._offset_from_row(row)
615             if count > remaining:
616                 count = remaining
617             row_offset = offset // self.row_size
618             # Mark the rows as being removed
619             self._remove_rows(subdir, filename, row_offset, row_offset + count)
620             remaining -= count
621             row += count
```

Listing B-10: `nilmdb/server/rocket.c`: "Rocket" low-level optimized data I/O module. The Rocket interface is used by the `bulkdata` interface in Listing B-9 to perform the lowest level of data transformation and disk I/O operations. It is a Python extension module, written in C, in order to maximize performance.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/server/rocket.c`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  #include <Python.h>
2  #include <structmember.h>
3  #include <endian.h>
4
5  #include <ctype.h>
6  #include <stdint.h>
7
8  #define __STDC_FORMAT_MACROS
9  #include <inttypes.h>
10
11 /* Values missing from stdint.h */
12 #define UINT8_MIN 0
13 #define UINT16_MIN 0
14 #define UINT32_MIN 0
15 #define UINT64_MIN 0
16
17 /* Marker values (if min == max, skip range check) */
18 #define FLOAT32_MIN 0
19 #define FLOAT32_MAX 0
20 #define FLOAT64_MIN 0
21 #define FLOAT64_MAX 0
22
23 typedef int64_t timestamp_t;
```

```

/* Somewhat arbitrary, just so we can use fixed sizes for strings
   etc. */
static const int MAX_LAYOUT_COUNT = 1024;

/* Error object and constants */
static PyObject *ParseError;
typedef enum {
        ERR_OTHER,
        ERR_NON_MONOTONIC,
        ERR_OUT_OF_INTERVAL,
} parseerror_code_t;
static void add_parseerror_codes(PyObject *module)
{
        PyModule_AddIntMacro(module, ERR_OTHER);
        PyModule_AddIntMacro(module, ERR_NON_MONOTONIC);
        PyModule_AddIntMacro(module, ERR_OUT_OF_INTERVAL);
}

/* Helpers to raise ParseErrors.  Use "return raise_str(...)" etc. */
static PyObject *raise_str(int line, int col, int code, const char *string)
{
        PyObject *o;
        o = Py_BuildValue("(iiis)", line, col, code, string);
        if (o != NULL) {
                PyErr_SetObject(ParseError, o);
                Py_DECREF(o);
        }
        return NULL;
}
static PyObject *raise_int(int line, int col, int code, int64_t num)
{
        PyObject *o;
        o = Py_BuildValue("(iiiL)", line, col, code, (long long)num);
        if (o != NULL) {
                PyErr_SetObject(ParseError, o);
                Py_DECREF(o);
        }
        return NULL;
}

/****
 * Layout and type helpers
 */
typedef union {
        int8_t i;
        uint8_t u;
} union8_t;
typedef union {
        int16_t i;
        uint16_t u;
} union16_t;
typedef union {
        int32_t i;
        uint32_t u;
        float f;
} union32_t;
typedef union {
        int64_t i;
        uint64_t u;
        double d;
} union64_t;

typedef enum {
        LAYOUT_TYPE_NONE,
        LAYOUT_TYPE_INT8,
        LAYOUT_TYPE_UINT8,
```

```
        LAYOUT_TYPE_INT16,
        LAYOUT_TYPE_UINT16,
        LAYOUT_TYPE_INT32,
        LAYOUT_TYPE_UINT32,
        LAYOUT_TYPE_INT64,
        LAYOUT_TYPE_UINT64,
        LAYOUT_TYPE_FLOAT32,
        LAYOUT_TYPE_FLOAT64,
} layout_type_t;

struct {
        char *string;
        layout_type_t layout;
        int size;
} type_lookup[] = {
        { "int8",    LAYOUT_TYPE_INT8,    1 },
        { "uint8",   LAYOUT_TYPE_UINT8,   1 },
        { "int16",   LAYOUT_TYPE_INT16,   2 },
        { "uint16",  LAYOUT_TYPE_UINT16,  2 },
        { "int32",   LAYOUT_TYPE_INT32,   4 },
        { "uint32",  LAYOUT_TYPE_UINT32,  4 },
        { "int64",   LAYOUT_TYPE_INT64,   8 },
        { "uint64",  LAYOUT_TYPE_UINT64,  8 },
        { "float32", LAYOUT_TYPE_FLOAT32, 4 },
        { "float64", LAYOUT_TYPE_FLOAT64, 8 },
        { NULL }
};

/****
 * Object definition, init, etc
 */

/* Rocket object */
typedef struct {
        PyObject_HEAD
        layout_type_t layout_type;
        int layout_count;
        int binary_size;
        FILE *file;
        int file_size;
} Rocket;

/* Dealloc / new */
static void Rocket_dealloc(Rocket *self)
{
        if (self->file) {
                fprintf(stderr, "rocket: file wasn't closed\n");
                fclose(self->file);
                self->file = NULL;
        }
        self->ob_type->tp_free((PyObject *)self);
}

static PyObject *Rocket_new(PyTypeObject *type, PyObject *args, PyObject *kwds)
{
        Rocket *self;

        self = (Rocket *)type->tp_alloc(type, 0);
        if (!self)
                return NULL;
        self->layout_type = LAYOUT_TYPE_NONE;
        self->layout_count = 0;
        self->binary_size = 0;
        self->file = NULL;
        self->file_size = -1;
        return (PyObject *)self;
}
```

```c

/* .__init__(layout, file) */
static int Rocket_init(Rocket *self, PyObject *args, PyObject *kwds)
{
        const char *layout, *path;
        static char *kwlist[] = { "layout", "file", NULL };
        if (!PyArg_ParseTupleAndKeywords(args, kwds, "sz", kwlist,
                                         &layout, &path))
                return -1;
        if (!layout)
                return -1;
        if (path) {
                if ((self->file = fopen(path, "a+b")) == NULL) {
                        PyErr_SetFromErrno(PyExc_OSError);
                        return -1;
                }
                self->file_size = -1;
        } else {
                self->file = NULL;
        }

        const char *under;
        char *tmp;
        under = strchr(layout, '_');
        if (!under) {
                PyErr_SetString(PyExc_ValueError, "no such layout: "
                                "badly formatted string");
                return -1;
        }
        self->layout_count = strtoul(under+1, &tmp, 10);
        if (self->layout_count < 1 || *tmp != '\0') {
                PyErr_SetString(PyExc_ValueError, "no such layout: "
                                "bad count");
                return -1;
        }
        if (self->layout_count >= MAX_LAYOUT_COUNT) {
                PyErr_SetString(PyExc_ValueError, "no such layout: "
                                "count too high");
                return -1;
        }

        int i;
        for (i = 0; type_lookup[i].string; i++)
                if (strncmp(layout, type_lookup[i].string, under-layout) == 0)
                        break;
        if (!type_lookup[i].string) {
                PyErr_SetString(PyExc_ValueError, "no such layout: "
                                "bad data type");
                return -1;
        }
        self->layout_type = type_lookup[i].layout;
        self->binary_size = 8 + (type_lookup[i].size * self->layout_count);

        return 0;
}

/* .close() */
static PyObject *Rocket_close(Rocket *self)
{
        if (self->file) {
                fclose(self->file);
                self->file = NULL;
        }
        Py_INCREF(Py_None);
        return Py_None;
}

```

```
/* .file_size property */
static PyObject *Rocket_get_file_size(Rocket *self)
{
        if (!self->file) {
                PyErr_SetString(PyExc_AttributeError, "no file");
                return NULL;
        }
        if (self->file_size < 0) {
                int oldpos;
                if (((oldpos = ftell(self->file)) < 0) ||
                    (fseek(self->file, 0, SEEK_END) < 0) ||
                    ((self->file_size = ftell(self->file)) < 0) ||
                    (fseek(self->file, oldpos, SEEK_SET) < 0)) {
                        PyErr_SetFromErrno(PyExc_OSError);
                        return NULL;
                }
        }
        return PyInt_FromLong(self->file_size);
}

/****
 * Append from string
 */
static inline long int strtoll10(const char *nptr, char **endptr) {
        return strtoll(nptr, endptr, 10);
}
static inline long int strtoull10(const char *nptr, char **endptr) {
        return strtoull(nptr, endptr, 10);
}

/* .append_string(count, data, offset, linenum, start, end, last_timestamp) */
static PyObject *Rocket_append_string(Rocket *self, PyObject *args)
{
        int count;
        const char *data;
        int offset;
        const char *linestart;
        int linenum;
        long long ll1, ll2, ll3;
        timestamp_t start;
        timestamp_t end;
        timestamp_t last_timestamp;

        int written = 0;
        char *endptr;
        union8_t t8;
        union16_t t16;
        union32_t t32;
        union64_t t64;
        int i;

        /* It would be nice to use 't#' instead of 's' for data,
           but we need the null termination for strto*.  If we had
           strnto* that took a length, we could use t# and not require
           a copy. */
        if (!PyArg_ParseTuple(args, "isiiLLL:append_string", &count,
                              &data, &offset, &linenum,
                              &ll1, &ll2, &ll3))
                return NULL;
        start = ll1;
        end = ll2;
        last_timestamp = ll3;

        /* Skip spaces, but don't skip over a newline. */
#define SKIP_BLANK(buf) do {                    \
        while (isspace(*buf)) {                 \
                if (*buf == '\n')               \
```

```
                        break;                  \
                buf++;                          \
        } } while(0)

        const char *buf = &data[offset];
        while (written < count && *buf)
        {
                linestart = buf;
                linenum++;

                /* Skip leading whitespace and commented lines */
                SKIP_BLANK(buf);
                if (*buf == '#') {
                        while (*buf && *buf != '\n')
                                buf++;
                        if (*buf)
                                buf++;
                        continue;
                }

                /* Extract timestamp */
                t64.i = strtoll(buf, &endptr, 10);
                if (endptr == buf || !isspace(*endptr)) {
                        /* Try parsing as a double instead */
                        t64.d = strtod(buf, &endptr);
                        if (endptr == buf)
                                goto bad_timestamp;
                        if (!isspace(*endptr))
                                goto cant_parse_value;
                        t64.i = round(t64.d);
                }
                if (t64.i <= last_timestamp)
                        return raise_int(linenum, buf - linestart + 1,
                                         ERR_NON_MONOTONIC, t64.i);
                last_timestamp = t64.i;
                if (t64.i < start || t64.i >= end)
                        return raise_int(linenum, buf - linestart + 1,
                                         ERR_OUT_OF_INTERVAL, t64.i);
                t64.u = le64toh(t64.u);
                if (fwrite(&t64.u, 8, 1, self->file) != 1)
                        goto err;
                buf = endptr;

                /* Parse all values in the line */
                switch (self->layout_type) {
#define CS(type, parsefunc, parsetype, realtype, disktype, letoh, bytes) \
                case LAYOUT_TYPE_##type:                                \
                        /* parse and write in a loop */                 \
                        for (i = 0; i < self->layout_count; i++) {      \
                                /* skip non-newlines */                 \
                                SKIP_BLANK(buf);                        \
                                if (*buf == '\n')                       \
                                        goto wrong_number_of_values;    \
                                /* parse number */                      \
                                parsetype = parsefunc(buf, &endptr);    \
                                if (*endptr && !isspace(*endptr))       \
                                        goto cant_parse_value;          \
                                /* check limits */                      \
                                if (type##_MIN != type##_MAX &&         \
                                    (parsetype < type##_MIN ||          \
                                     parsetype > type##_MAX))           \
                                        goto value_out_of_range;        \
                                /* convert to disk representation */    \
                                realtype = parsetype;                   \
                                disktype = letoh(disktype);             \
                                /* write it */                          \
                                if (fwrite(&disktype, bytes,            \
```

```
                                        1, self->file) != 1)         \
                                    goto err;                        \
                            /* advance buf */                        \
                            buf = endptr;                            \
                    }                                                \
                    /* Skip trailing whitespace and comments */      \
                    SKIP_BLANK(buf);                                 \
                    if (*buf == '#')                                 \
                            while (*buf && *buf != '\n')             \
                                    buf++;                           \
                    if (*buf == '\n')                                \
                            buf++;                                   \
                    else if (*buf != '\0')                           \
                            goto extra_data_on_line;                 \
                    break

                    CS(INT8,   strtoll10,  t64.i, t8.i,  t8.u,         , 1);
                    CS(UINT8,  strtoull10, t64.u, t8.u,  t8.u,         , 1);
                    CS(INT16,  strtoll10,  t64.i, t16.i, t16.u, le16toh, 2);
                    CS(UINT16, strtoull10, t64.u, t16.u, t16.u, le16toh, 2);
                    CS(INT32,  strtoll10,  t64.i, t32.i, t32.u, le32toh, 4);
                    CS(UINT32, strtoull10, t64.u, t32.u, t32.u, le32toh, 4);
                    CS(INT64,  strtoll10,  t64.i, t64.i, t64.u, le64toh, 8);
                    CS(UINT64, strtoull10, t64.u, t64.u, t64.u, le64toh, 8);
                    CS(FLOAT32, strtod,   t64.d, t32.f, t32.u, le32toh, 4);
                    CS(FLOAT64, strtod,   t64.d, t64.d, t64.u, le64toh, 8);
#undef CS
                default:
                        PyErr_SetString(PyExc_TypeError, "unknown type");
                        return NULL;
                }

                /* Done this line */
                written++;
        }

        fflush(self->file);

        /* Build return value and return */
        offset = buf - data;
        PyObject *o;
        o = Py_BuildValue("(iiLi)", written, offset,
                          (long long)last_timestamp, linenum);
        return o;
err:
        PyErr_SetFromErrno(PyExc_OSError);
        return NULL;
bad_timestamp:
        return raise_str(linenum, buf - linestart + 1,
                         ERR_OTHER, "bad timestamp");
cant_parse_value:
        return raise_str(linenum, buf - linestart + 1,
                         ERR_OTHER, "can't parse value");
wrong_number_of_values:
        return raise_str(linenum, buf - linestart + 1,
                         ERR_OTHER, "wrong number of values");
value_out_of_range:
        return raise_str(linenum, buf - linestart + 1,
                         ERR_OTHER, "value out of range");
extra_data_on_line:
        return raise_str(linenum, buf - linestart + 1,
                         ERR_OTHER, "extra data on line");
}

/****
 * Append from binary data
 */
```

```
426
427 /* .append_binary(count, data, offset, linenum, start, end, last_timestamp) */
428 static PyObject *Rocket_append_binary(Rocket *self, PyObject *args)
429 {
430         int count;
431         const uint8_t *data;
432         int data_len;
433         int linenum;
434         int offset;
435         long long ll1, ll2, ll3;
436         timestamp_t start;
437         timestamp_t end;
438         timestamp_t last_timestamp;
439
440         if (!PyArg_ParseTuple(args, "it#iiLLL:append_binary",
441                               &count, &data, &data_len, &offset,
442                               &linenum, &ll1, &ll2, &ll3))
443                 return NULL;
444         start = ll1;
445         end = ll2;
446         last_timestamp = ll3;
447
448         /* Advance to offset */
449         if (offset > data_len)
450                 return raise_str(0, 0, ERR_OTHER, "bad offset");
451         data += offset;
452         data_len -= offset;
453
454         /* Figure out max number of rows to insert */
455         int rows = data_len / self->binary_size;
456         if (rows > count)
457                 rows = count;
458
459         /* Check timestamps */
460         timestamp_t ts;
461         int i;
462         for (i = 0; i < rows; i++) {
463                 /* Read raw timestamp, byteswap if needed */
464                 memcpy(&ts, &data[i * self->binary_size], 8);
465                 ts = le64toh(ts);
466
467                 /* Check limits */
468                 if (ts <= last_timestamp)
469                         return raise_int(i, 0, ERR_NON_MONOTONIC, ts);
470                 last_timestamp = ts;
471                 if (ts < start || ts >= end)
472                         return raise_int(i, 0, ERR_OUT_OF_INTERVAL, ts);
473         }
474
475         /* Write binary data */
476         if (fwrite(data, self->binary_size, rows, self->file) != rows) {
477                 PyErr_SetFromErrno(PyExc_OSError);
478                 return NULL;
479         }
480         fflush(self->file);
481
482         /* Build return value and return */
483         PyObject *o;
484         o = Py_BuildValue("(iiLi)", rows, offset + rows * self->binary_size,
485                           (long long)last_timestamp, linenum);
486         return o;
487 }
488
489 /****
490  * Extract to string
491  */
492
```

```
static PyObject *Rocket_extract_string(Rocket *self, PyObject *args)
{
        long count;
        long offset;

        if (!PyArg_ParseTuple(args, "ll", &offset, &count))
                return NULL;
        if (!self->file) {
                PyErr_SetString(PyExc_Exception, "no file");
                return NULL;
        }
        /* Seek to target location */
        if (fseek(self->file, offset, SEEK_SET) < 0) {
                PyErr_SetFromErrno(PyExc_OSError);
                return NULL;
        }

        char *str = NULL, *new;
        long len_alloc = 0;
        long len = 0;
        int ret;

        /* min space free in string (and the maximum length of one
           line); this is generous */
        const int min_free = 32 * MAX_LAYOUT_COUNT;

        /* how much to allocate at once */
        const int alloc_size = 1048576;

        int row, i;
        union8_t t8;
        union16_t t16;
        union32_t t32;
        union64_t t64;
        for (row = 0; row < count; row++) {
                /* Make sure there's space for a line */
                if ((len_alloc - len) < min_free) {
                        /* grow by 1 meg at a time */
                        len_alloc += alloc_size;
                        new = realloc(str, len_alloc);
                        if (new == NULL)
                                goto err;
                        str = new;
                }

                /* Read and print timestamp */
                if (fread(&t64.u, 8, 1, self->file) != 1)
                        goto err;
                t64.u = le64toh(t64.u);
                ret = sprintf(&str[len], "%" PRId64, t64.i);
                if (ret <= 0)
                        goto err;
                len += ret;

                /* Read and print values */
                switch (self->layout_type) {
#define CASE(type, fmt, fmttype, disktype, letoh, bytes)                \
                case LAYOUT_TYPE_##type:                                \
                        /* read and format in a loop */                 \
                        for (i = 0; i < self->layout_count; i++) {      \
                                if (fread(&disktype, bytes,             \
                                          1, self->file) != 1)          \
                                        goto err;                       \
                                disktype = letoh(disktype);             \
                                ret = sprintf(&str[len], " " fmt,       \
                                              fmttype);                 \
                                if (ret <= 0)                           \
```

```
                                        goto err;                       \
                                len += ret;                             \
                        }                                               \
                        break
                        CASE(INT8,   "%" PRId8,  t8.i,  t8.u,         , 1);
                        CASE(UINT8,  "%" PRIu8,  t8.u,  t8.u,         , 1);
                        CASE(INT16,  "%" PRId16, t16.i, t16.u, le16toh, 2);
                        CASE(UINT16, "%" PRIu16, t16.u, t16.u, le16toh, 2);
                        CASE(INT32,  "%" PRId32, t32.i, t32.u, le32toh, 4);
                        CASE(UINT32, "%" PRIu32, t32.u, t32.u, le32toh, 4);
                        CASE(INT64,  "%" PRId64, t64.i, t64.u, le64toh, 8);
                        CASE(UINT64, "%" PRIu64, t64.u, t64.u, le64toh, 8);
                        /* These next two are a bit debatable.  floats
                           are 6-9 significant figures, so we print 7.
                           Doubles are 15-19, so we print 17.  This is
                           similar to the old prep format for float32.
                        */
                        CASE(FLOAT32, "%.6e",  t32.f, t32.u, le32toh, 4);
                        CASE(FLOAT64, "%.16e", t64.d, t64.u, le64toh, 8);
#undef CASE
                default:
                        PyErr_SetString(PyExc_TypeError, "unknown type");
                        if (str) free(str);
                        return NULL;
                }
                str[len++] = '\n';
        }

        PyObject *pystr = PyString_FromStringAndSize(str, len);
        free(str);
        return pystr;
err:
        if (str) free(str);
        PyErr_SetFromErrno(PyExc_OSError);
        return NULL;
}

/****
 * Extract to binary string containing raw little-endian binary data
 */
static PyObject *Rocket_extract_binary(Rocket *self, PyObject *args)
{
        long count;
        long offset;

        if (!PyArg_ParseTuple(args, "ll", &offset, &count))
                return NULL;
        if (!self->file) {
                PyErr_SetString(PyExc_Exception, "no file");
                return NULL;
        }
        /* Seek to target location */
        if (fseek(self->file, offset, SEEK_SET) < 0) {
                PyErr_SetFromErrno(PyExc_OSError);
                return NULL;
        }

        uint8_t *str;
        int len = count * self->binary_size;
        str = malloc(len);
        if (str == NULL) {
                PyErr_SetFromErrno(PyExc_OSError);
                return NULL;
        }

        /* Data in the file is already in the desired little-endian
           binary format, so just read it directly. */
```

```
        if (fread(str, self->binary_size, count, self->file) != count) {
                free(str);
                PyErr_SetFromErrno(PyExc_OSError);
                return NULL;
        }

        PyObject *pystr = PyBytes_FromStringAndSize((char *)str, len);
        free(str);
        return pystr;
}

/****
 * Extract timestamp
 */
static PyObject *Rocket_extract_timestamp(Rocket *self, PyObject *args)
{
        long offset;
        union64_t t64;
        if (!PyArg_ParseTuple(args, "l", &offset))
                return NULL;
        if (!self->file) {
                PyErr_SetString(PyExc_Exception, "no file");
                return NULL;
        }

        /* Seek to target location and read timestamp */
        if ((fseek(self->file, offset, SEEK_SET) < 0) ||
            (fread(&t64.u, 8, 1, self->file) != 1)) {
                PyErr_SetFromErrno(PyExc_OSError);
                return NULL;
        }

        /* Convert and return */
        t64.u = le64toh(t64.u);
        return Py_BuildValue("L", (long long)t64.i);
}

/****
 * Module and type setup
 */

static PyGetSetDef Rocket_getsetters[] = {
        { "file_size", (getter)Rocket_get_file_size, NULL,
          "file size in bytes", NULL },
        { NULL },
};

static PyMemberDef Rocket_members[] = {
        { "binary_size", T_INT, offsetof(Rocket, binary_size), 0,
          "binary size per row" },
        { NULL },
};

static PyMethodDef Rocket_methods[] = {
        { "close",
          (PyCFunction)Rocket_close, METH_NOARGS,
          "close(self)\n\n"
          "Close file handle" },

        { "append_string",
          (PyCFunction)Rocket_append_string, METH_VARARGS,
          "append_string(self, count, data, offset, line, start, end, ts)\n\n"
          "Parse string and append data.\n"
          "\n"
          "  count: maximum number of rows to add\n"
          "  data: string data\n"
          "  offset: byte offset into data to start parsing\n"
```

```
694             "  line: current line number of data\n"
695             "  start: starting timestamp for interval\n"
696             "  end: end timestamp for interval\n"
697             "  ts: last timestamp that was previously parsed\n"
698             "\n"
699             "Raises ParseError if timestamps are non-monotonic, outside\n"
700             "the start/end interval etc.\n"
701             "\n"
702             "On success, return a tuple:\n"
703             "  added_rows: how many rows were added from the file\n"
704             "  data_offset: current offset into the data string\n"
705             "  last_timestamp: last timestamp we parsed\n"
706             "  linenum: current line number" },
707
708         { "append_binary",
709           (PyCFunction)Rocket_append_binary, METH_VARARGS,
710           "append_binary(self, count, data, offset, line, start, end, ts)\n\n"
711           "Append binary data, which must match the data layout.\n"
712           "\n"
713           "  count: maximum number of rows to add\n"
714           "  data: binary data\n"
715           "  offset: byte offset into data to start adding\n"
716           "  line: current line number (unused)\n"
717           "  start: starting timestamp for interval\n"
718           "  end: end timestamp for interval\n"
719           "  ts: last timestamp that was previously parsed\n"
720           "\n"
721           "Raises ParseError if timestamps are non-monotonic, outside\n"
722           "the start/end interval etc.\n"
723           "\n"
724           "On success, return a tuple:\n"
725           "  added_rows: how many rows were added from the file\n"
726           "  data_offset: current offset into the data string\n"
727           "  last_timestamp: last timestamp we parsed\n"
728           "  linenum: current line number (copied from argument)" },
729
730         { "extract_string",
731           (PyCFunction)Rocket_extract_string, METH_VARARGS,
732           "extract_string(self, offset, count)\n\n"
733           "Extract count rows of data from the file at offset offset.\n"
734           "Return an ascii formatted string according to the layout" },
735
736         { "extract_binary",
737           (PyCFunction)Rocket_extract_binary, METH_VARARGS,
738           "extract_binary(self, offset, count)\n\n"
739           "Extract count rows of data from the file at offset offset.\n"
740           "Return a raw binary string of data matching the data layout." },
741
742         { "extract_timestamp",
743           (PyCFunction)Rocket_extract_timestamp, METH_VARARGS,
744           "extract_timestamp(self, offset)\n\n"
745           "Extract a single timestamp from the file" },
746
747         { NULL },
748 };
749
750 static PyTypeObject RocketType = {
751         PyObject_HEAD_INIT(NULL)
752
753         .tp_name        = "rocket.Rocket",
754         .tp_basicsize   = sizeof(Rocket),
755         .tp_flags       = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
756
757         .tp_new         = Rocket_new,
758         .tp_dealloc     = (destructor)Rocket_dealloc,
759         .tp_init        = (initproc)Rocket_init,
760         .tp_methods     = Rocket_methods,
```

```
761         .tp_members     = Rocket_members,
762         .tp_getset      = Rocket_getsetters,
763
764         .tp_doc         = ("rocket.Rocket(layout, file)\n\n"
765                            "C implementation of the \"rocket\" data parsing\n"
766                            "interface, which translates between the binary\n"
767                            "format on disk and the ASCII or Python list\n"
768                            "format used when communicating with the rest of\n"
769                            "the system.")
770 };
771
772 static PyMethodDef module_methods[] = {
773         { NULL },
774 };
775
776 PyMODINIT_FUNC
777 initrocket(void)
778 {
779         PyObject *module;
780
781         RocketType.tp_new = PyType_GenericNew;
782         if (PyType_Ready(&RocketType) < 0)
783                 return;
784
785         module = Py_InitModule3("rocket", module_methods,
786                                 "Rocket data parsing and formatting module");
787         Py_INCREF(&RocketType);
788         PyModule_AddObject(module, "Rocket", (PyObject *)&RocketType);
789
790         ParseError = PyErr_NewException("rocket.ParseError", NULL, NULL);
791         Py_INCREF(ParseError);
792         PyModule_AddObject(module, "ParseError", ParseError);
793         add_parseerror_codes(module);
794
795         return;
796 }
```

B.2 Database Check and Repair

Listing B-11: `nilmdb/scripts/nilmdb_fsck.py`: Entry point for database consistency check and repair tool.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/scripts/nilmdb_fsck.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  #!/usr/bin/python
2
3  import nilmdb.fsck
4  import argparse
5  import os
6  import sys
7
8  def main():
9      """Main entry point for the 'nilmdb-fsck' command line script"""
10
11     parser = argparse.ArgumentParser(
12         description = 'Check database consistency',
13         formatter_class = argparse.ArgumentDefaultsHelpFormatter,
14         version = nilmdb.__version__)
```

```
    parser.add_argument("-f", "--fix", action="store_true",
                        default=False, help = 'Fix errors when possible '
                        '(which may involve removing data)')
    parser.add_argument("-n", "--no-data", action="store_true",
                        default=False, help = 'Skip the slow full-data check')
    parser.add_argument('database', help = 'Database directory')
    args = parser.parse_args()

    nilmdb.fsck.Fsck(args.database, args.fix).check(skip_data = args.no_data)

if __name__ == "__main__":
    main()
```

Listing B-12: `nilmdb/fsck/fsck.py`: Database consistency check and repair routines. Verifies database health and correctness, which may be needed after an improper shutdown or other corruption. Some errors are corrected automatically if requested by the user.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/fsck/fsck.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# -*- coding: utf-8 -*-

"""Check database consistency, with some ability to fix problems.
This should be able to fix cases where a database gets corrupted due
to unexpected system shutdown, and detect other cases that may cause
NilmDB to return errors when trying to manipulate the database."""

import nilmdb.utils
import nilmdb.server
import nilmdb.client.numpyclient
from nilmdb.utils.interval import IntervalError
from nilmdb.server.interval import Interval, IntervalSet
from nilmdb.utils.printf import *
from nilmdb.utils.time import timestamp_to_string

from collections import defaultdict
import sqlite3
import os
import sys
import progressbar
import re
import time
import shutil
import cPickle as pickle
import numpy

class FsckError(Exception):
    def __init__(self, msg = "", *args):
        if args:
            msg = sprintf(msg, *args)
        Exception.__init__(self, msg)
class FixableFsckError(FsckError):
    def __init__(self, msg = "", *args):
        if args:
            msg = sprintf(msg, *args)
        FsckError.__init__(self, "%s\nThis may be fixable with \"--fix\".", msg)
class RetryFsck(FsckError):
    pass

def log(format, *args):
```

```
    printf(format, *args)

def err(format, *args):
    fprintf(sys.stderr, format, *args)

# Decorator that retries a function if it returns a specific value
def retry_if_raised(exc, message = None, max_retries = 100):
    def f1(func):
        def f2(*args, **kwargs):
            for n in range(max_retries):
                try:
                    return func(*args, **kwargs)
                except exc as e:
                    if message:
                        log("%s\n\n", message)
            raise Exception("Max number of retries (%d) exceeded; giving up")
        return f2
    return f1

class Progress(object):
    def __init__(self, maxval):
        self.bar = progressbar.ProgressBar(
            maxval = maxval,
            widgets = [ progressbar.Percentage(), ' ',
                        progressbar.Bar(), ' ',
                        progressbar.ETA() ])
        if self.bar.term_width == 0:
            self.bar.term_width = 75
    def __enter__(self):
        self.bar.start()
        self.last_update = 0
        return self
    def __exit__(self, exc_type, exc_value, traceback):
        if exc_type is None:
            self.bar.finish()
        else:
            printf("\n")
    def update(self, val):
        self.bar.update(val)

class Fsck(object):

    def __init__(self, path, fix = False):
        self.basepath = path
        self.sqlpath = os.path.join(path, "data.sql")
        self.bulkpath = os.path.join(path, "data")
        self.bulklock = os.path.join(path, "data.lock")
        self.fix = fix

    ### Main checks

    @retry_if_raised(RetryFsck, "Something was fixed: restarting fsck")
    def check(self, skip_data = False):
        self.bulk = None
        self.sql = None
        try:
            self.check_paths()
            self.check_sql()
            self.check_streams()
            self.check_intervals()
            if skip_data:
                log("skipped data check\n")
            else:
                self.check_data()
        finally:
            if self.bulk:
                self.bulk.close()
```

```
            if self.sql:
                self.sql.commit()
                self.sql.close()
        log("ok\n")

    ### Check basic path structure

    def check_paths(self):
        log("checking paths\n")
        if self.bulk:
            self.bulk.close()
        if not os.path.isfile(self.sqlpath):
            raise FsckError("SQL database missing (%s)", self.sqlpath)
        if not os.path.isdir(self.bulkpath):
            raise FsckError("Bulk data directory missing (%s)", self.bulkpath)
        with open(self.bulklock, "w") as lockfile:
            if not nilmdb.utils.lock.exclusive_lock(lockfile):
                raise FsckError('Database already locked by another process\n'
                                'Make sure all other processes that might be '
                                'using the database are stopped.\n'
                                'Restarting apache will cause it to unlock '
                                'the db until a request is received.')
            # unlocked immediately
        self.bulk = nilmdb.server.bulkdata.BulkData(self.basepath)

    ### Check SQL database health

    def check_sql(self):
        log("checking sqlite database\n")

        self.sql = sqlite3.connect(self.sqlpath)
        with self.sql:
            cur = self.sql.cursor()
            ver = cur.execute("PRAGMA user_version").fetchone()[0]
            good = max(nilmdb.server.nilmdb._sql_schema_updates.keys())
            if ver != good:
                raise FsckError("database version %d too old, should be %d",
                                ver, good)
            self.stream_path = {}
            self.stream_layout = {}
            log("  loading paths\n")
            result = cur.execute("SELECT id, path, layout FROM streams")
            for r in result:
                if r[0] in self.stream_path:
                    raise FsckError("duplicated ID %d in stream IDs", r[0])
                self.stream_path[r[0]] = r[1]
                self.stream_layout[r[0]] = r[2]

            log("  loading intervals\n")
            self.stream_interval = defaultdict(list)
            result = cur.execute("SELECT stream_id, start_time, end_time, "
                                 "start_pos, end_pos FROM ranges "
                                 "ORDER BY start_time")
            for r in result:
                if r[0] not in self.stream_path:
                    raise FsckError("interval ID %d not in streams", k)
                self.stream_interval[r[0]].append((r[1], r[2], r[3], r[4]))

            log("  loading metadata\n")
            self.stream_meta = defaultdict(dict)
            result = cur.execute("SELECT stream_id, key, value FROM metadata")
            for r in result:
                if r[0] not in self.stream_path:
                    raise FsckError("metadata ID %d not in streams", k)
                if r[1] in self.stream_meta[r[0]]:
                    raise FsckError("duplicate metadata key '%s' for stream %d",
                                    r[1], r[0])
```

```
175                 self.stream_meta[r[0]][r[1]] = r[2]
176
177         ### Check streams and basic interval overlap
178
179         def check_streams(self):
180             ids = self.stream_path.keys()
181             log("checking %s streams\n", "{:,d}".format(len(ids)))
182             with Progress(len(ids)) as pbar:
183                 for i, sid in enumerate(ids):
184                     pbar.update(i)
185                     path = self.stream_path[sid]
186
187                     # unique path, valid layout
188                     if self.stream_path.values().count(path) != 1:
189                         raise FsckError("duplicated path %s", path)
190                     layout = self.stream_layout[sid].split('_')[0]
191                     if layout not in ('int8', 'int16', 'int32', 'int64',
192                                       'uint8', 'uint16', 'uint32', 'uint64',
193                                       'float32', 'float64'):
194                         raise FsckError("bad layout %s for %s", layout, path)
195                     count = int(self.stream_layout[sid].split('_')[1])
196                     if count < 1 or count > 1024:
197                         raise FsckError("bad count %d for %s", count, path)
198
199                     # must exist in bulkdata
200                     bulk = self.bulkpath + path
201                     if not os.path.isdir(bulk):
202                         raise FsckError("%s: missing bulkdata dir", path)
203                     if not nilmdb.server.bulkdata.Table.exists(bulk):
204                         raise FsckError("%s: bad bulkdata table", path)
205
206                     # intervals don't overlap.  Abuse IntervalSet to check
207                     # for intervals in file positions, too.
208                     timeiset = IntervalSet()
209                     posiset = IntervalSet()
210                     for (stime, etime, spos, epos) in self.stream_interval[sid]:
211                         new = Interval(stime, etime)
212                         try:
213                             timeiset += new
214                         except IntervalError:
215                             raise FsckError("%s: overlap in intervals:\n"
216                                             "set: %s\nnew: %s",
217                                             path, str(timeiset), str(new))
218                         if spos != epos:
219                             new = Interval(spos, epos)
220                             try:
221                                 posiset += new
222                             except IntervalError:
223                                 raise FsckError("%s: overlap in file offsets:\n"
224                                                 "set: %s\nnew: %s",
225                                                 path, str(posiset), str(new))
226
227                     # check bulkdata
228                     self.check_bulkdata(sid, path, bulk)
229
230                     # Check that we can open bulkdata
231                     try:
232                         tab = None
233                         try:
234                             tab = nilmdb.server.bulkdata.Table(bulk)
235                         except Exception as e:
236                             raise FsckError("%s: can't open bulkdata: %s",
237                                             path, str(e))
238                     finally:
239                         if tab:
240                             tab.close()
241
```

```
    ### Check that bulkdata is good enough to be opened

    @retry_if_raised(RetryFsck)
    def check_bulkdata(self, sid, path, bulk):
        with open(os.path.join(bulk, "_format"), "rb") as f:
            fmt = pickle.load(f)
        if fmt["version"] != 3:
            raise FsckError("%s: bad or unsupported bulkdata version %d",
                            path, fmt["version"])
        row_per_file = int(fmt["rows_per_file"])
        files_per_dir = int(fmt["files_per_dir"])
        layout = fmt["layout"]
        if layout != self.stream_layout[sid]:
            raise FsckError("%s: layout mismatch %s != %s", path,
                            layout, self.stream_layout[sid])

        # Every file should have a size that's the multiple of the row size
        rkt = nilmdb.server.rocket.Rocket(layout, None)
        row_size = rkt.binary_size
        rkt.close()

        # Find all directories
        regex = re.compile("^[0-9a-f]{4,}$")
        subdirs = sorted(filter(regex.search, os.listdir(bulk)),
                         key = lambda x: int(x, 16), reverse = True)
        for subdir in subdirs:
            # Find all files in that dir
            subpath = os.path.join(bulk, subdir)
            files = filter(regex.search, os.listdir(subpath))
            if not files:
                self.fix_empty_subdir(subpath)
                raise RetryFsck
            # Verify that their size is a multiple of the row size
            for filename in files:
                filepath = os.path.join(subpath, filename)
                offset = os.path.getsize(filepath)
                if offset % row_size:
                    self.fix_bad_filesize(path, filepath, offset, row_size)

    def fix_empty_subdir(self, subpath):
        msg = sprintf("bulkdata path %s is missing data files", subpath)
        if not self.fix:
            raise FixableFsckError(msg)
        # Try to fix it by just deleting whatever is present,
        # as long as it's only ".removed" files.
        err("\n%s\n", msg)
        for fn in os.listdir(subpath):
            if not fn.endswith(".removed"):
                raise FsckError("can't fix automatically: please manually "
                                "remove the file %s and try again",
                                os.path.join(subpath, fn))
        # Remove the whole thing
        err("Removing empty subpath\n")
        shutil.rmtree(subpath)
        raise RetryFsck

    def fix_bad_filesize(self, path, filepath, offset, row_size):
        extra = offset % row_size
        msg = sprintf("%s: size of file %s (%d) is not a multiple" +
                      " of row size (%d): %d extra bytes present",
                      path, filepath, offset, row_size, extra)
        if not self.fix:
            raise FixableFsckError(msg)
        # Try to fix it by just truncating the file
        err("\n%s\n", msg)
        newsize = offset - extra
        err("Truncating file to %d bytes and retrying\n", newsize)
```

```
309             with open(filepath, "r+b") as f:
310                 f.truncate(newsize)
311                 raise RetryFsck
312 
313         ### Check interval endpoints
314 
315         def check_intervals(self):
316             total_ints = sum(len(x) for x in self.stream_interval.values())
317             log("checking %s intervals\n", "{:,d}".format(total_ints))
318             done = 0
319             with Progress(total_ints) as pbar:
320                 for sid in self.stream_interval:
321                     try:
322                         bulk = self.bulkpath + self.stream_path[sid]
323                         tab = nilmdb.server.bulkdata.Table(bulk)
324                         def update(x):
325                             pbar.update(done + x)
326                         ints = self.stream_interval[sid]
327                         done += self.check_table_intervals(sid, ints, tab, update)
328                     finally:
329                         tab.close()
330 
331         def check_table_intervals(self, sid, ints, tab, update):
332             # look in the table to make sure we can pick out the interval's
333             # endpoints
334             path = self.stream_path[sid]
335             tab.file_open.cache_remove_all()
336             for (i, intv) in enumerate(ints):
337                 update(i)
338                 (stime, etime, spos, epos) = intv
339                 if spos == epos and spos >= 0 and spos <= tab.nrows:
340                     continue
341                 try:
342                     srow = tab[spos]
343                     erow = tab[epos-1]
344                 except Exception as e:
345                     self.fix_bad_interval(sid, intv, tab, str(e))
346                     raise RetryFsck
347             return len(ints)
348 
349         def fix_bad_interval(self, sid, intv, tab, msg):
350             path = self.stream_path[sid]
351             msg = sprintf("%s: interval %s error accessing rows: %s",
352                           path, str(intv), str(msg))
353             if not self.fix:
354                 raise FixableFsckError(msg)
355             err("\n%s\n", msg)
356 
357             (stime, etime, spos, epos) = intv
358             # If it's just that the end pos is more than the number of rows
359             # in the table, lower end pos and truncate interval time too.
360             if spos < tab.nrows and epos >= tab.nrows:
361                 err("end position is past endrows, but it can be truncated\n")
362                 err("old end: time %d, pos %d\n", etime, epos)
363                 new_epos = tab.nrows
364                 new_etime = tab[new_epos-1] + 1
365                 err("new end: time %d, pos %d\n", new_etime, new_epos)
366                 if stime < new_etime:
367                     # Change it in SQL
368                     with self.sql:
369                         cur = self.sql.cursor()
370                         cur.execute("UPDATE ranges SET end_time=?, end_pos=? "
371                                     "WHERE stream_id=? AND start_time=? AND "
372                                     "end_time=? AND start_pos=? AND end_pos=?",
373                                     (new_etime, new_epos, sid, stime, etime,
374                                      spos, epos))
375                         if cur.rowcount != 1:
```

```
                    raise FsckError("failed to fix SQL database")
                raise RetryFsck
            err("actually it can't be truncated; times are bad too")

        # Otherwise, the only hope is to delete the interval entirely.
        err("*** Deleting the entire interval from SQL.\n")
        err("This may leave stale data on disk.  To fix that, copy all\n")
        err("data from this stream to a new stream, then remove all data\n")
        err("from and destroy %s.\n", path)
        with self.sql:
            cur = self.sql.cursor()
            cur.execute("DELETE FROM ranges WHERE "
                        "stream_id=? AND start_time=? AND "
                        "end_time=? AND start_pos=? AND end_pos=?",
                        (sid, stime, etime, spos, epos))
            if cur.rowcount != 1:
                raise FsckError("failed to remove interval")
        raise RetryFsck

    ### Check data in each interval

    def check_data(self):
        total_rows = sum(sum((y[3] - y[2]) for y in x)
                         for x in self.stream_interval.values())
        log("checking %s rows of data\n", "{:,d}".format(total_rows))
        done = 0
        with Progress(total_rows) as pbar:
            for sid in self.stream_interval:
                try:
                    bulk = self.bulkpath + self.stream_path[sid]
                    tab = nilmdb.server.bulkdata.Table(bulk)
                    def update(x):
                        pbar.update(done + x)
                    ints = self.stream_interval[sid]
                    done += self.check_table_data(sid, ints, tab, update)
                finally:
                    tab.close()

    def check_table_data(self, sid, ints, tab, update):
        # Pull out all of the interval's data and verify that it's
        # monotonic.
        maxrows = 100000
        path = self.stream_path[sid]
        layout = self.stream_layout[sid]
        dtype = nilmdb.client.numpyclient.layout_to_dtype(layout)
        tab.file_open.cache_remove_all()
        done = 0
        for intv in ints:
            last_ts = None
            (stime, etime, spos, epos) = intv
            if spos == epos:
                continue
            for start in xrange(*slice(spos, epos, maxrows).indices(epos)):
                stop = min(start + maxrows, epos)
                count = stop - start
                # Get raw data, convert to NumPy arary
                try:
                    raw = tab.get_data(start, stop, binary = True)
                    data = numpy.fromstring(raw, dtype)
                except Exception as e:
                    raise FsckError("%s: failed to grab rows %d through %d: %s",
                                    path, start, stop, repr(e))

                # Verify that timestamps are monotonic
                if (numpy.diff(data['timestamp']) <= 0).any():
                    raise FsckError("%s: non-monotonic timestamp(s) in rows "
                                    "%d through %d", path, start, stop)
```

```
443                 first_ts = data['timestamp'][0]
444                 if last_ts is not None and first_ts <= last_ts:
445                     raise FsckError("%s: first interval timestamp %d is not "
446                                     "greater than the previous last interval "
447                                     "timestamp %d, at row %d",
448                                     path, first_ts, last_ts, start)
449                 last_ts = data['timestamp'][-1]
450
451                 # These are probably fixable, by removing the offending
452                 # intervals.  But I'm not going to bother implementing
453                 # that yet.
454
455                 # Done
456                 done += count
457                 update(done)
458         return done
```

B.3 Client Library

Listing B-13: `nilmdb/client/__init__.py`: Client library module init.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/client/__init__.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1  """nilmdb.client"""
2
3  from nilmdb.client.client import Client
4  from nilmdb.client.errors import ClientError, ServerError, Error
```

Listing B-14: `nilmdb/client/client.py`: Main client library implementation. This includes the `Client` class and the `StreamInserter` context manager, and is generally focused towards text-based data.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/client/client.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1  # -*- coding: utf-8 -*-
2
3  """Class for performing HTTP client requests via libcurl"""
4
5  import nilmdb.utils
6  import nilmdb.client.httpclient
7  from nilmdb.client.errors import ClientError
8
9  import time
10 import simplejson as json
11 import contextlib
12
13 from nilmdb.utils.time import timestamp_to_string, string_to_timestamp
14
15 def extract_timestamp(line):
16     """Extract just the timestamp from a line of data text"""
17     return string_to_timestamp(line.split()[0])
18
19 class Client(object):
```

```
    """Main client interface to the Nilm database."""

    def __init__(self, url, post_json = False):
        """Initialize client with given URL.  If post_json is true,
        POST requests are sent with Content-Type 'application/json'
        instead of the default 'x-www-form-urlencoded'."""
        self.http = nilmdb.client.httpclient.HTTPClient(url, post_json)
        self.post_json = post_json

    # __enter__/__exit__ allow this class to be a context manager
    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self.close()

    def _json_post_param(self, data):
        """Return compact json-encoded version of parameter"""
        if self.post_json:
            # If we're posting as JSON, we don't need to encode it further here
            return data
        return json.dumps(data, separators=(',',':'))

    def close(self):
        """Close the connection; safe to call multiple times"""
        self.http.close()

    def geturl(self):
        """Return the URL we're using"""
        return self.http.baseurl

    def version(self):
        """Return server version"""
        return self.http.get("version")

    def dbinfo(self):
        """Return server database info (path, size, free space)
        as a dictionary."""
        return self.http.get("dbinfo")

    def stream_list(self, path = None, layout = None, extended = False):
        """Return a sorted list of [path, layout] lists.  If 'path' or
        'layout' are specified, only return streams that match those
        exact values.  If 'extended' is True, the returned lists have
        extended info, e.g.: [path, layout, extent_min, extent_max,
        total_rows, total_seconds."""
        params = {}
        if path is not None:
            params["path"] = path
        if layout is not None:
            params["layout"] = layout
        if extended:
            params["extended"] = 1
        streams = self.http.get("stream/list", params)
        return nilmdb.utils.sort.sort_human(streams, key = lambda s: s[0])

    def stream_get_metadata(self, path, keys = None):
        """Get stream metadata"""
        params = { "path": path }
        if keys is not None:
            params["key"] = keys
        return self.http.get("stream/get_metadata", params)

    def stream_set_metadata(self, path, data):
        """Set stream metadata from a dictionary, replacing all existing
        metadata."""
        params = {
```

```
            "path": path,
            "data": self._json_post_param(data)
            }
        return self.http.post("stream/set_metadata", params)

    def stream_update_metadata(self, path, data):
        """Update stream metadata from a dictionary"""
        params = {
            "path": path,
            "data": self._json_post_param(data)
            }
        return self.http.post("stream/update_metadata", params)

    def stream_create(self, path, layout):
        """Create a new stream"""
        params = { "path": path,
                   "layout" : layout }
        return self.http.post("stream/create", params)

    def stream_destroy(self, path):
        """Delete stream.  Fails if any data is still present."""
        params = { "path": path }
        return self.http.post("stream/destroy", params)

    def stream_rename(self, oldpath, newpath):
        """Rename a stream."""
        params = { "oldpath": oldpath,
                   "newpath": newpath }
        return self.http.post("stream/rename", params)

    def stream_remove(self, path, start = None, end = None):
        """Remove data from the specified time range"""
        params = {
            "path": path
        }
        if start is not None:
            params["start"] = timestamp_to_string(start)
        if end is not None:
            params["end"] = timestamp_to_string(end)
        total = 0
        for count in self.http.post_gen("stream/remove", params):
            total += int(count)
        return total

    @contextlib.contextmanager
    def stream_insert_context(self, path, start = None, end = None):
        """Return a context manager that allows data to be efficiently
        inserted into a stream in a piecewise manner.  Data is
        provided as ASCII lines, and is aggregated and sent to the
        server in larger or smaller chunks as necessary.  Data lines
        must match the database layout for the given path, and end
        with a newline.

        Example:
          with client.stream_insert_context('/path', start, end) as ctx:
            ctx.insert('1234567890.0 1 2 3 4\\n')
            ctx.insert('1234567891.0 1 2 3 4\\n')

        For more details, see help for nilmdb.client.client.StreamInserter

        This may make multiple requests to the server, if the data is
        large enough or enough time has passed between insertions.
        """
        ctx = StreamInserter(self, path, start, end)
        yield ctx
        ctx.finalize()
        ctx.destroy()
```

```
154
155     def stream_insert(self, path, data, start = None, end = None):
156         """Insert rows of data into a stream.  data should be a string
157         or iterable that provides ASCII data that matches the database
158         layout for path.  Data is passed through stream_insert_context,
159         so it will be broken into reasonably-sized chunks and
160         start/end will be deduced if missing."""
161         with self.stream_insert_context(path, start, end) as ctx:
162             if isinstance(data, basestring):
163                 ctx.insert(data)
164             else:
165                 for chunk in data:
166                     ctx.insert(chunk)
167         return ctx.last_response
168
169     def stream_insert_block(self, path, data, start, end, binary = False):
170         """Insert a single fixed block of data into the stream.  It is
171         sent directly to the server in one block with no further
172         processing.
173
174         If 'binary' is True, provide raw binary data in little-endian
175         format matching the path layout, including an int64 timestamp.
176         Otherwise, provide ASCII data matching the layout."""
177         params = {
178             "path": path,
179             "start": timestamp_to_string(start),
180             "end": timestamp_to_string(end),
181         }
182         if binary:
183             params["binary"] = 1
184         return self.http.put("stream/insert", data, params, binary = binary)
185
186     def stream_intervals(self, path, start = None, end = None, diffpath = None):
187         """
188         Return a generator that yields each stream interval.
189
190         If 'diffpath' is not None, yields only interval ranges that are
191         present in 'path' but not in 'diffpath'.
192         """
193         params = {
194             "path": path
195         }
196         if diffpath is not None:
197             params["diffpath"] = diffpath
198         if start is not None:
199             params["start"] = timestamp_to_string(start)
200         if end is not None:
201             params["end"] = timestamp_to_string(end)
202         return self.http.get_gen("stream/intervals", params)
203
204     def stream_extract(self, path, start = None, end = None,
205                        count = False, markup = False, binary = False):
206         """
207         Extract data from a stream.  Returns a generator that yields
208         lines of ASCII-formatted data that matches the database
209         layout for the given path.
210
211         If 'count' is True, return a count of matching data points
212         rather than the actual data.  The output format is unchanged.
213
214         If 'markup' is True, include comments in the returned data
215         that indicate interval starts and ends.
216
217         If 'binary' is True, return chunks of raw binary data, rather
218         than lines of ASCII-formatted data.  Raw binary data is
219         little-endian and matches the database types (including an
220         int64 timestamp).
```

```
        """
        params = {
            "path": path,
        }
        if start is not None:
            params["start"] = timestamp_to_string(start)
        if end is not None:
            params["end"] = timestamp_to_string(end)
        if count:
            params["count"] = 1
        if markup:
            params["markup"] = 1
        if binary:
            params["binary"] = 1
        return self.http.get_gen("stream/extract", params, binary = binary)

    def stream_count(self, path, start = None, end = None):
        """
        Return the number of rows of data in the stream that satisfy
        the given timestamps.
        """
        counts = list(self.stream_extract(path, start, end, count = True))
        return int(counts[0])

class StreamInserter(object):
    """Object returned by stream_insert_context() that manages
    the insertion of rows of data into a particular path.

    The basic data flow is that we are filling a contiguous interval
    on the server, with no gaps, that extends from timestamp 'start'
    to timestamp 'end'.  Data timestamps satisfy 'start <= t < end'.

    Data is provided to .insert() as ASCII formatted data separated by
    newlines.  The chunks of data passed to .insert() do not need to
    match up with the newlines; less or more than one line can be passed.

    1. The first inserted line begins a new interval that starts at
    'start'.  If 'start' is not given, it is deduced from the first
    line's timestamp.

    2. Subsequent lines go into the same contiguous interval.  As lines
    are inserted, this routine may make multiple insertion requests to
    the server, but will structure the timestamps to leave no gaps.

    3. The current contiguous interval can be completed by manually
    calling .finalize(), which the context manager will also do
    automatically.  This will send any remaining data to the server,
    using the 'end' timestamp to end the interval.  If no 'end'
    was provided, it is deduced from the last timestamp seen,
    plus a small delta.

    After a .finalize(), inserting new data goes back to step 1.

    .update_start() can be called before step 1 to change the start
    time for the interval.  .update_end() can be called before step 3
    to change the end time for the interval.
    """

    # See design.md for a discussion of how much data to send.  This
    # is a soft limit -- we might send up to twice as much or so
    _max_data = 2 * 1024 * 1024
    _max_data_after_send = 64 * 1024

    def __init__(self, client, path, start, end):
        """'client' is the client object.  'path' is the database
        path to insert to.  'start' and 'end' are used for the first
        contiguous interval and may be None."""
```

```
        self.last_response = None

        self._client = client
        self._path = path

        # Start and end for the overall contiguous interval we're
        # filling
        self._interval_start = start
        self._interval_end = end

        # Current data we're building up to send.  Each string
        # goes into the array, and gets joined all at once.
        self._block_data = []
        self._block_len = 0

        self.destroyed = False

    def destroy(self):
        """Ensure this object can't be used again without raising
        an error"""
        def error(*args, **kwargs):
            raise Exception("don't reuse this context object")
        self._send_block = self.insert = self.finalize = self.send = error

    def insert(self, data):
        """Insert a chunk of ASCII formatted data in string form.  The
        overall data must consist of lines terminated by '\\n'."""
        length = len(data)
        maxdata = self._max_data

        if length > maxdata:
            # This could make our buffer more than twice what we
            # wanted to send, so split it up.  This is a bit
            # inefficient, but the user really shouldn't be providing
            # this much data at once.
            for cut in range(0, length, maxdata):
                self.insert(data[cut:(cut + maxdata)])
            return

        # Append this string to our list
        self._block_data.append(data)
        self._block_len += length

        # Send the block once we have enough data
        if self._block_len >= maxdata:
            self._send_block(final = False)
            if self._block_len >= self._max_data_after_send: # pragma: no cover
                raise ValueError("too much data left over after trying"
                                 " to send intermediate block; is it"
                                 " missing newlines or malformed?")

    def update_start(self, start):
        """Update the start time for the next contiguous interval.
        Call this before starting to insert data for a new interval,
        for example, after .finalize()"""
        self._interval_start = start

    def update_end(self, end):
        """Update the end time for the current contiguous interval.
        Call this before .finalize()"""
        self._interval_end = end

    def finalize(self):
        """Stop filling the current contiguous interval.
        All outstanding data will be sent, and the interval end
        time of the interval will be taken from the 'end' argument
        used when initializing this class, or the most recent
```

```
355             value passed to update_end(), or the last timestamp plus
356             a small epsilon value if no other endpoint was provided.
357 
358             If more data is inserted after a finalize(), it will become
359             part of a new interval and there may be a gap left in-between."""
360             self._send_block(final = True)
361 
362         def send(self):
363             """Send any data that we might have buffered up.  Does not affect
364             any other treatment of timestamps or endpoints."""
365             self._send_block(final = False)
366 
367         def _get_first_noncomment(self, block):
368             """Return the (start, end) indices of the first full line in
369             block that isn't a comment, or raise IndexError if
370             there isn't one."""
371             start = 0
372             while True:
373                 end = block.find('\n', start)
374                 if end < 0:
375                     raise IndexError
376                 if block[start] != '#':
377                     return (start, (end + 1))
378                 start = end + 1
379 
380         def _get_last_noncomment(self, block):
381             """Return the (start, end) indices of the last full line in
382             block[:length] that isn't a comment, or raise IndexError if
383             there isn't one."""
384             end = block.rfind('\n')
385             if end <= 0:
386                 raise IndexError
387             while True:
388                 start = block.rfind('\n', 0, end)
389                 if block[start + 1] != '#':
390                     return ((start + 1), end)
391                 if start == -1:
392                     raise IndexError
393                 end = start
394 
395         def _send_block(self, final = False):
396             """Send data currently in the block.  The data sent will
397             consist of full lines only, so some might be left over."""
398             # Build the full string to send
399             block = "".join(self._block_data)
400 
401             start_ts = self._interval_start
402             if start_ts is None:
403                 # Pull start from the first line
404                 try:
405                     (spos, epos) = self._get_first_noncomment(block)
406                     start_ts = extract_timestamp(block[spos:epos])
407                 except (ValueError, IndexError):
408                     pass # no timestamp is OK, if we have no data
409 
410             if final:
411                 # For a final block, it must end in a newline, and the
412                 # ending timestamp is either the user-provided end,
413                 # or the timestamp of the last line plus epsilon.
414                 end_ts = self._interval_end
415                 try:
416                     if block[-1] != '\n':
417                         raise ValueError("final block didn't end with a newline")
418                     if end_ts is None:
419                         (spos, epos) = self._get_last_noncomment(block)
420                         end_ts = extract_timestamp(block[spos:epos])
421                         end_ts += nilmdb.utils.time.epsilon
```

```
422                 except (ValueError, IndexError):
423                     pass # no timestamp is OK, if we have no data
424                 self._block_data = []
425                 self._block_len = 0
426 
427                 # Next block is completely fresh
428                 self._interval_start = None
429                 self._interval_end = None
430             else:
431                 # An intermediate block, e.g. "line1\nline2\nline3\nline4"
432                 # We need to save "line3\nline4" for the next block, and
433                 # use the timestamp from "line3" as the ending timestamp
434                 # for this one.
435                 try:
436                     (spos, epos) = self._get_last_noncomment(block)
437                     end_ts = extract_timestamp(block[spos:epos])
438                 except (ValueError, IndexError):
439                     # If we found no timestamp, give up; we could send this
440                     # block later when we have more data.
441                     return
442                 if spos == 0:
443                     # Not enough data to send an intermediate block
444                     return
445                 if self._interval_end is not None and end_ts > self._interval_end:
446                     # User gave us bad endpoints; send it anyway, and let
447                     # the server complain so that the error is the same
448                     # as if we hadn't done this chunking.
449                     end_ts = self._interval_end
450                 self._block_data = [ block[spos:] ]
451                 self._block_len = (epos - spos)
452                 block = block[:spos]
453 
454                 # Next block continues where this one ended
455                 self._interval_start = end_ts
456 
457             # Double check endpoints
458             if (start_ts is None or end_ts is None) or (start_ts == end_ts):
459                 # If the block has no non-comment lines, it's OK
460                 try:
461                     self._get_first_noncomment(block)
462                 except IndexError:
463                     return
464                 raise ClientError("have data to send, but no start/end times")
465 
466             # Send it
467             self.last_response = self._client.stream_insert_block(
468                 self._path, block, start_ts, end_ts, binary = False)
469 
470             return
```

Listing B-15: `nilmdb/client/numpyclient.py`: Additional client library with NumPy array support. The `NumpyClient` and `StreamInserterNumpy` classes are subclasses of `Client` and `StreamInserter` that support efficient data insertion and retrieval using NumPy arrays.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/client/numpyclient.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1 # -*- coding: utf-8 -*-
2 
```

```
"""Provide a NumpyClient class that is based on normal Client, but has
additional methods for extracting and inserting data via Numpy arrays."""

import nilmdb.utils
import nilmdb.client.client
import nilmdb.client.httpclient
from nilmdb.client.errors import ClientError

import contextlib
from nilmdb.utils.time import timestamp_to_string, string_to_timestamp

import numpy
import cStringIO

def layout_to_dtype(layout):
    ltype = layout.split('_')[0]
    lcount = int(layout.split('_')[1])
    if ltype.startswith('int'):
        atype = '<i' + str(int(ltype[3:]) / 8)
    elif ltype.startswith('uint'):
        atype = '<u' + str(int(ltype[4:]) / 8)
    elif ltype.startswith('float'):
        atype = '<f' + str(int(ltype[5:]) / 8)
    else:
        raise ValueError("bad layout")
    return numpy.dtype([('timestamp', '<i8'), ('data', atype, lcount)])

class NumpyClient(nilmdb.client.client.Client):
    """Subclass of nilmdb.client.Client that adds additional methods for
    extracting and inserting data via Numpy arrays."""

    def _get_dtype(self, path, layout):
        if layout is None:
            streams = self.stream_list(path)
            if len(streams) != 1:
                raise ClientError("can't get layout for path: " + path)
            layout = streams[0][1]
        return layout_to_dtype(layout)

    def stream_extract_numpy(self, path, start = None, end = None,
                             layout = None, maxrows = 100000,
                             structured = False):
        """
        Extract data from a stream.  Returns a generator that yields
        Numpy arrays of up to 'maxrows' of data each.

        If 'layout' is None, it is read using stream_info.

        If 'structured' is False, all data is converted to float64
        and returned in a flat 2D array.  Otherwise, data is returned
        as a structured dtype in a 1D array.
        """
        dtype = self._get_dtype(path, layout)

        def to_numpy(data):
            a = numpy.fromstring(data, dtype)
            if structured:
                return a
            return numpy.c_[a['timestamp'], a['data']]

        chunks = []
        total_len = 0
        maxsize = dtype.itemsize * maxrows
        for data in self.stream_extract(path, start, end, binary = True):
            # Add this block of binary data
            chunks.append(data)
            total_len += len(data)
```

```
70
71              # See if we have enough to make the requested Numpy array
72              while total_len >= maxsize:
73                  assembled = "".join(chunks)
74                  total_len -= maxsize
75                  chunks = [ assembled[maxsize:] ]
76                  block = assembled[:maxsize]
77                  yield to_numpy(block)
78
79          if total_len:
80              yield to_numpy("".join(chunks))
81
82      @contextlib.contextmanager
83      def stream_insert_numpy_context(self, path, start = None, end = None,
84                                      layout = None):
85          """Return a context manager that allows data to be efficiently
86          inserted into a stream in a piecewise manner.  Data is
87          provided as Numpy arrays, and is aggregated and sent to the
88          server in larger or smaller chunks as necessary.  Data format
89          must match the database layout for the given path.
90
91          For more details, see help for
92          nilmdb.client.numpyclient.StreamInserterNumpy
93
94          If 'layout' is not None, use it as the layout rather than
95          querying the database.
96          """
97          dtype = self._get_dtype(path, layout)
98          ctx = StreamInserterNumpy(self, path, start, end, dtype)
99          yield ctx
100         ctx.finalize()
101         ctx.destroy()
102
103     def stream_insert_numpy(self, path, data, start = None, end = None,
104                             layout = None):
105         """Insert data into a stream.  data should be a Numpy array
106         which will be passed through stream_insert_numpy_context to
107         break it into chunks etc.  See the help for that function
108         for details."""
109         with self.stream_insert_numpy_context(path, start, end, layout) as ctx:
110             if isinstance(data, numpy.ndarray):
111                 ctx.insert(data)
112             else:
113                 for chunk in data:
114                     ctx.insert(chunk)
115         return ctx.last_response
116
117 class StreamInserterNumpy(nilmdb.client.client.StreamInserter):
118     """Object returned by stream_insert_numpy_context() that manages
119     the insertion of rows of data into a particular path.
120
121     See help for nilmdb.client.client.StreamInserter for details.
122     The only difference is that, instead of ASCII formatted data,
123     this context manager can take Numpy arrays, which are either
124     structured (1D with complex dtype) or flat (2D with simple dtype).
125     """
126
127     # Soft limit of how many bytes to send per HTTP request.
128     _max_data = 2 * 1024 * 1024
129
130     def __init__(self, client, path, start, end, dtype):
131         """
132         'client' is the client object.  'path' is the database path
133         to insert to.  'start' and 'end' are used for the first
134         contiguous interval and may be None.  'dtype' is the Numpy
135         dtype for this stream.
136         """
```

```
137         super(StreamInserterNumpy, self).__init__(client, path, start, end)
138         self._dtype = dtype
139 
140         # Max rows to send at once
141         self._max_rows = self._max_data // self._dtype.itemsize
142 
143         # List of the current arrays we're building up to send
144         self._block_arrays = []
145         self._block_rows = 0
146 
147     def insert(self, array):
148         """Insert Numpy data, which must match the layout type."""
149         if type(array) != numpy.ndarray:
150             array = numpy.array(array)
151         if array.ndim == 1:
152             # Already a structured array; just verify the type
153             if array.dtype != self._dtype:
154                 raise ValueError("wrong dtype for 1D (structured) array")
155         elif array.ndim == 2:
156             # Convert to structured array
157             sarray = numpy.zeros(array.shape[0], dtype=self._dtype)
158             try:
159                 sarray['timestamp'] = array[:,0]
160                 # Need the squeeze in case sarray['data'] is 1 dimensional
161                 sarray['data'] = numpy.squeeze(array[:,1:])
162             except (IndexError, ValueError):
163                 raise ValueError("wrong number of fields for this data type")
164             array = sarray
165         else:
166             raise ValueError("wrong number of dimensions in array")
167 
168         length = len(array)
169         maxrows = self._max_rows
170 
171         if length == 0:
172             return
173         if length > maxrows:
174             # This is more than twice what we wanted to send, so split
175             # it up.  This is a bit inefficient, but the user really
176             # shouldn't be providing this much data at once.
177             for cut in range(0, length, maxrows):
178                 self.insert(array[cut:(cut + maxrows)])
179             return
180 
181         # Add this array to our list
182         self._block_arrays.append(array)
183         self._block_rows += length
184 
185         # Send if it's too long
186         if self._block_rows >= maxrows:
187             self._send_block(final = False)
188 
189     def _send_block(self, final = False):
190         """Send the data current stored up.  One row might be left
191         over if we need its timestamp saved."""
192 
193         # Build the full array to send
194         if self._block_rows == 0:
195             array = numpy.zeros(0, dtype = self._dtype)
196         else:
197             array = numpy.hstack(self._block_arrays)
198 
199         # Get starting timestamp
200         start_ts = self._interval_start
201         if start_ts is None:
202             # Pull start from the first row
203             try:
```

```
                start_ts = array['timestamp'][0]
            except IndexError:
                pass # no timestamp is OK, if we have no data

        # Get ending timestamp
        if final:
            # For a final block, the timestamp is either the
            # user-provided end, or the timestamp of the last line
            # plus epsilon.
            end_ts = self._interval_end
            if end_ts is None:
                try:
                    end_ts = array['timestamp'][-1]
                    end_ts += nilmdb.utils.time.epsilon
                except IndexError:
                    pass # no timestamp is OK, if we have no data
            self._block_arrays = []
            self._block_rows = 0

            # Next block is completely fresh
            self._interval_start = None
            self._interval_end = None
        else:
            # An intermediate block.  We need to save the last row
            # for the next block, and use its timestamp as the ending
            # timestamp for this one.
            if len(array) < 2:
                # Not enough data to send an intermediate block
                return
            end_ts = array['timestamp'][-1]
            if self._interval_end is not None and end_ts > self._interval_end:
                # User gave us bad endpoints; send it anyway, and let
                # the server complain so that the error is the same
                # as if we hadn't done this chunking.
                end_ts = self._interval_end
            self._block_arrays = [ array[-1:] ]
            self._block_rows = 1
            array = array[:-1]

            # Next block continues where this one ended
            self._interval_start = end_ts

        # If we have no endpoints, or equal endpoints, it's OK as long
        # as there's no data to send
        if (start_ts is None or end_ts is None) or (start_ts == end_ts):
            if len(array) == 0:
                return
            raise ClientError("have data to send, but invalid start/end times")

        # Send it
        data = array.tostring()
        self.last_response = self._client.stream_insert_block(
            self._path, data, start_ts, end_ts, binary = True)

        return
```

Listing B-16: `nilmdb/client/httpclient.py`: Low level HTTP client tools. This is used by the client classes and is implemented using the Requests library.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/client/httpclient.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
"""HTTP client library"""

import nilmdb.utils
from nilmdb.client.errors import ClientError, ServerError, Error

import simplejson as json
import urlparse
import requests

class HTTPClient(object):
    """Class to manage and perform HTTP requests from the client"""
    def __init__(self, baseurl = "", post_json = False, verify_ssl = True):
        """If baseurl is supplied, all other functions that take
        a URL can be given a relative URL instead."""
        # Verify / clean up URL
        reparsed = urlparse.urlparse(baseurl).geturl()
        if '://' not in reparsed:
            reparsed = urlparse.urlparse("http://" + baseurl).geturl()
        self.baseurl = reparsed.rstrip('/') + '/'

        # Build Requests session object, enable SSL verification
        self.verify_ssl = verify_ssl
        self.session = requests.Session()
        self.session.verify = True

        # Saved response, so that tests can verify a few things.
        self._last_response = {}

        # Whether to send application/json POST bodies (versus
        # x-www-form-urlencoded)
        self.post_json = post_json

    def _handle_error(self, url, code, body):
        # Default variables for exception.  We use the entire body as
        # the default message, in case we can't extract it from a JSON
        # response.
        args = { "url" : url,
                 "status" : str(code),
                 "message" : body,
                 "traceback" : None }
        try:
            # Fill with server-provided data if we can
            jsonerror = json.loads(body)
            args["status"] = jsonerror["status"]
            args["message"] = jsonerror["message"]
            args["traceback"] = jsonerror["traceback"]
        except Exception: # pragma: no cover
            pass
        if code >= 400 and code <= 499:
            raise ClientError(**args)
        else: # pragma: no cover
            if code >= 500 and code <= 599:
                if args["message"] is None:
                    args["message"] = ("(no message; try disabling " +
                                       "response.stream option in " +
                                       "nilmdb.server for better debugging)")
                raise ServerError(**args)
```

```
            else:
                raise Error(**args)

    def close(self):
        self.session.close()

    def _do_req(self, method, url, query_data, body_data, stream, headers):
        url = urlparse.urljoin(self.baseurl, url)
        try:
            response = self.session.request(method, url,
                                            params = query_data,
                                            data = body_data,
                                            stream = stream,
                                            headers = headers,
                                            verify = self.verify_ssl)
        except requests.RequestException as e:
            raise ServerError(status = "502 Error", url = url,
                              message = str(e.message))
        if response.status_code != 200:
            self._handle_error(url, response.status_code, response.content)
        self._last_response = response
        if response.headers["content-type"] in ("application/json",
                                                "application/x-json-stream"):
            return (response, True)
        else:
            return (response, False)

    # Normal versions that return data directly
    def _req(self, method, url, query = None, body = None, headers = None):
        """
        Make a request and return the body data as a string or parsed
        JSON object, or raise an error if it contained an error.
        """
        (response, isjson) = self._do_req(method, url, query, body,
                                          stream = False, headers = headers)
        if isjson:
            return json.loads(response.content)
        return response.content

    def get(self, url, params = None):
        """Simple GET (parameters in URL)"""
        return self._req("GET", url, params, None)

    def post(self, url, params = None):
        """Simple POST (parameters in body)"""
        if self.post_json:
            return self._req("POST", url, None,
                             json.dumps(params),
                             { 'Content-type': 'application/json' })
        else:
            return self._req("POST", url, None, params)

    def put(self, url, data, params = None, binary = False):
        """Simple PUT (parameters in URL, data in body)"""
        if binary:
            h = { 'Content-type': 'application/octet-stream' }
        else:
            h = { 'Content-type': 'text/plain; charset=utf-8' }
        return self._req("PUT", url, query = params, body = data, headers = h)

    # Generator versions that return data one line at a time.
    def _req_gen(self, method, url, query = None, body = None,
                 headers = None, binary = False):
        """
        Make a request and return a generator that gives back strings
        or JSON decoded lines of the body data, or raise an error if
        it contained an eror.
```

```
        """
        (response, isjson) = self._do_req(method, url, query, body,
                                          stream = True, headers = headers)

        # Like the iter_lines function in Requests, but only splits on
        # the specified line ending.
        def lines(source, ending):
            pending = None
            for chunk in source:
                if pending is not None:
                    chunk = pending + chunk
                tmp = chunk.split(ending)
                lines = tmp[:-1]
                if chunk.endswith(ending):
                    pending = None
                else:
                    pending = tmp[-1]
                for line in lines:
                    yield line
            if pending is not None: # pragma: no cover (missing newline)
                yield pending

        # Yield the chunks or lines as requested
        if binary:
            for chunk in response.iter_content(chunk_size = 65536):
                yield chunk
        elif isjson:
            for line in lines(response.iter_content(chunk_size = 1),
                              ending = '\r\n'):
                yield json.loads(line)
        else:
            for line in lines(response.iter_content(chunk_size = 65536),
                              ending = '\n'):
                yield line

    def get_gen(self, url, params = None, binary = False):
        """Simple GET (parameters in URL) returning a generator"""
        return self._req_gen("GET", url, params, binary = binary)

    def post_gen(self, url, params = None):
        """Simple POST (parameters in body) returning a generator"""
        if self.post_json:
            return self._req_gen("POST", url, None,
                                 json.dumps(params),
                                 { 'Content-type': 'application/json' })
        else:
            return self._req_gen("POST", url, None, params)

    # Not much use for a POST or PUT generator, since they don't
    # return much data.
```

B.4 Command Line Interface

Listing B-17: `nilmdb/scripts/nilmtool.py`: Main entry point for the `nilmtool` command line tool.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/scripts/nilmtool.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
#!/usr/bin/python

import nilmdb.cmdline

def main():
    """Main entry point for the 'nilmtool' command line script"""
    nilmdb.cmdline.Cmdline().run()

if __name__ == "__main__":
    main()
```

Listing B-18: `nilmdb/cmdline/cmdline.py`: Basic framework for command line processing. This class sets up the main command line parser, handles common options, and provides hooks to pass control to specific subcommands.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/cmdline/cmdline.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
"""Command line client functionality"""

import nilmdb.client

from nilmdb.utils.printf import *
from nilmdb.utils import datetime_tz
import nilmdb.utils.time

import sys
import os
import argparse
from argparse import ArgumentDefaultsHelpFormatter as def_form
import signal

try: # pragma: no cover
    import argcomplete
except ImportError: # pragma: no cover
    argcomplete = None

# Valid subcommands.  Defined in separate files just to break
# things up -- they're still called with Cmdline as self.
subcommands = [ "help", "info", "create", "rename", "list", "intervals",
                "metadata", "insert", "extract", "remove", "destroy" ]

# Import the subcommand modules
subcmd_mods = {}
for cmd in subcommands:
    subcmd_mods[cmd] = __import__("nilmdb.cmdline." + cmd, fromlist = [ cmd ])

```

```
class JimArgumentParser(argparse.ArgumentParser):
    def parse_args(self, args=None, namespace=None):
        # Look for --version anywhere and change it to just "nilmtool
        # --version".  This makes "nilmtool cmd --version" work, which
        # is needed by help2man.
        if "--version" in (args or sys.argv[1:]):
            args = [ "--version" ]
        return argparse.ArgumentParser.parse_args(self, args, namespace)

    def error(self, message):
        self.print_usage(sys.stderr)
        self.exit(2, sprintf("error: %s\n", message))

class Complete(object): # pragma: no cover
    # Completion helpers, for using argcomplete (see
    # extras/nilmtool-bash-completion.sh)
    def escape(self, s):
        quote_chars = [ "\\", "\"", "'", " " ]
        for char in quote_chars:
            s = s.replace(char, "\\" + char)
        return s

    def none(self, prefix, parsed_args, **kwargs):
        return []
    rate = none
    time = none
    url = none

    def path(self, prefix, parsed_args, **kwargs):
        client = nilmdb.client.Client(parsed_args.url)
        return ( self.escape(s[0])
                 for s in client.stream_list()
                 if s[0].startswith(prefix) )

    def layout(self, prefix, parsed_args, **kwargs):
        types = [ "int8", "int16", "int32", "int64",
                  "uint8", "uint16", "uint32", "uint64",
                  "float32", "float64" ]
        layouts = []
        for i in range(1,10):
            layouts.extend([(t + "_" + str(i)) for t in types])
        return ( l for l in layouts if l.startswith(prefix) )

    def meta_key(self, prefix, parsed_args, **kwargs):
        return (kv.split('=')[0] for kv
                in self.meta_keyval(prefix, parsed_args, **kwargs))

    def meta_keyval(self, prefix, parsed_args, **kwargs):
        client = nilmdb.client.Client(parsed_args.url)
        path = parsed_args.path
        if not path:
            return []
        results = []
        # prefix comes in as UTF-8, but results need to be Unicode,
        # weird.  Still doesn't work in all cases, but that's bugs in
        # argcomplete.
        prefix = nilmdb.utils.unicode.decode(prefix)
        for (k,v) in client.stream_get_metadata(path).iteritems():
            kv = self.escape(k + '=' + v)
            if kv.startswith(prefix):
                results.append(kv)
        return results

class Cmdline(object):

    def __init__(self, argv = None):
        self.argv = argv or sys.argv[1:]
```

```
        try:
            # Assume command line arguments are encoded with stdin's encoding,
            # and reverse it.  Won't be needed in Python 3, but for now..
            self.argv = [ x.decode(sys.stdin.encoding) for x in self.argv ]
        except Exception: # pragma: no cover
            pass
        self.client = None
        self.def_url = os.environ.get("NILMDB_URL", "http://localhost/nilmdb/")
        self.subcmd = {}
        self.complete = Complete()

    def arg_time(self, toparse):
        """Parse a time string argument"""
        try:
            return nilmdb.utils.time.parse_time(toparse)
        except ValueError as e:
            raise argparse.ArgumentTypeError(sprintf("%s \"%s\"",
                                                     str(e), toparse))

    # Set up the parser
    def parser_setup(self):
        self.parser = JimArgumentParser(add_help = False,
                                        formatter_class = def_form)

        group = self.parser.add_argument_group("General options")
        group.add_argument("-h", "--help", action='help',
                           help='show this help message and exit')
        group.add_argument("-v", "--version", action="version",
                           version = nilmdb.__version__)

        group = self.parser.add_argument_group("Server")
        group.add_argument("-u", "--url", action="store",
                           default=self.def_url,
                           help="NilmDB server URL (default: %(default)s)"
                           ).completer = self.complete.url

        sub = self.parser.add_subparsers(
            title="Commands", dest="command",
            description="Use 'help command' or 'command --help' for more "
            "details on a particular command.")

        # Set up subcommands (defined in separate files)
        for cmd in subcommands:
            self.subcmd[cmd] = subcmd_mods[cmd].setup(self, sub)

    def die(self, formatstr, *args):
        fprintf(sys.stderr, formatstr + "\n", *args)
        if self.client:
            self.client.close()
        sys.exit(-1)

    def run(self):
        # Set SIGPIPE to its default handler -- we don't need Python
        # to catch it for us.
        try:
            signal.signal(signal.SIGPIPE, signal.SIG_DFL)
        except ValueError: # pragma: no cover
            pass

        # Clear cached timezone, so that we can pick up timezone changes
        # while running this from the test suite.
        datetime_tz._localtz = None

        # Run parser
        self.parser_setup()
        if argcomplete: # pragma: no cover
            argcomplete.autocomplete(self.parser)
```

```
164         self.args = self.parser.parse_args(self.argv)
165
166         # Run arg verify handler if there is one
167         if "verify" in self.args:
168             self.args.verify(self)
169
170         self.client = nilmdb.client.Client(self.args.url)
171
172         # Make a test connection to make sure things work,
173         # unless the particular command requests that we don't.
174         if "no_test_connect" not in self.args:
175             try:
176                 server_version = self.client.version()
177             except nilmdb.client.Error as e:
178                 self.die("error connecting to server: %s", str(e))
179
180         # Now dispatch client request to appropriate function.  Parser
181         # should have ensured that we don't have any unknown commands
182         # here.
183         retval = self.args.handler(self) or 0
184
185         self.client.close()
186         sys.exit(retval)
```

Listing B-19: `nilmdb/cmdline/create.py`: Handler for subcommand `create`: create new empty streams.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/create.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1  from nilmdb.utils.printf import *
2  import nilmdb.client
3
4  from argparse import RawDescriptionHelpFormatter as raw_form
5
6  def setup(self, sub):
7      cmd = sub.add_parser("create", help="Create a new stream",
8                           formatter_class = raw_form,
9                           description="""
10 Create a new empty stream at the specified path and with the specified
11 layout type.
12
13 Layout types are of the format: type_count
14
15   'type' is a data type like 'float32', 'float64', 'uint16', 'int32', etc.
16
17   'count' is the number of columns of this type.
18
19   For example, 'float32_8' means the data for this stream has 8 columns of
20   32-bit floating point values.
21 """)
22     cmd.set_defaults(handler = cmd_create)
23     group = cmd.add_argument_group("Required arguments")
24     group.add_argument("path",
25                        help="Path (in database) of new stream, e.g. /foo/bar",
26                        ).completer = self.complete.path
27     group.add_argument("layout",
28                        help="Layout type for new stream, e.g. float32_8",
29                        ).completer = self.complete.layout
30     return cmd
31
32 def cmd_create(self):
```

```
33      """Create new stream"""
34      try:
35          self.client.stream_create(self.args.path, self.args.layout)
36      except nilmdb.client.ClientError as e:
37          self.die("error creating stream: %s", str(e))
```

Listing B-20: `nilmdb/cmdline/destroy.py`: Handler for subcommand `destroy`: delete stream and data.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/destroy.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   from nilmdb.utils.printf import *
2   import nilmdb.client
3   import fnmatch
4
5   from argparse import ArgumentDefaultsHelpFormatter as def_form
6
7   def setup(self, sub):
8       cmd = sub.add_parser("destroy", help="Delete a stream and all data",
9                            formatter_class = def_form,
10                           description="""
11                           Destroy the stream at the specified path.
12                           The stream must be empty.  All metadata
13                           related to the stream is permanently deleted.
14
15                           Wildcards and multiple paths are supported.
16                           """)
17      cmd.set_defaults(handler = cmd_destroy)
18      group = cmd.add_argument_group("Options")
19      group.add_argument("-R", "--remove", action="store_true",
20                         help="Remove all data before destroying stream")
21      group.add_argument("-q", "--quiet", action="store_true",
22                         help="Don't display names when destroying "
23                         "multiple paths")
24      group = cmd.add_argument_group("Required arguments")
25      group.add_argument("path", nargs='+',
26                         help="Path of the stream to delete, e.g. /foo/bar/*",
27                         ).completer = self.complete.path
28      return cmd
29
30  def cmd_destroy(self):
31      """Destroy stream"""
32      streams = [ s[0] for s in self.client.stream_list() ]
33      paths = []
34      for path in self.args.path:
35          new = fnmatch.filter(streams, path)
36          if not new:
37              self.die("error: no stream matched path: %s", path)
38          paths.extend(new)
39
40      for path in paths:
41          if not self.args.quiet and len(paths) > 1:
42              printf("Destroying %s\n", path)
43
44          try:
45              if self.args.remove:
46                  count = self.client.stream_remove(path)
47              self.client.stream_destroy(path)
48          except nilmdb.client.ClientError as e:
49              self.die("error destroying stream: %s", str(e))
```

Listing B-21: `nilmdb/cmdline/extract.py`: Handler for subcommand `extract`: extract data from a stream.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/extract.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
from __future__ import print_function
from nilmdb.utils.printf import *
import nilmdb.client
import sys

def setup(self, sub):
    cmd = sub.add_parser("extract", help="Extract data",
                         description="""
                         Extract data from a stream.
                         """)
    cmd.set_defaults(verify = cmd_extract_verify,
                     handler = cmd_extract)

    group = cmd.add_argument_group("Data selection")
    group.add_argument("path",
                       help="Path of stream, e.g. /foo/bar",
                       ).completer = self.complete.path
    group.add_argument("-s", "--start", required=True,
                       metavar="TIME", type=self.arg_time,
                       help="Starting timestamp (free-form, inclusive)",
                       ).completer = self.complete.time
    group.add_argument("-e", "--end", required=True,
                       metavar="TIME", type=self.arg_time,
                       help="Ending timestamp (free-form, noninclusive)",
                       ).completer = self.complete.time

    group = cmd.add_argument_group("Output format")
    group.add_argument("-B", "--binary", action="store_true",
                       help="Raw binary output")
    group.add_argument("-b", "--bare", action="store_true",
                       help="Exclude timestamps from output lines")
    group.add_argument("-a", "--annotate", action="store_true",
                       help="Include comments with some information "
                       "about the stream")
    group.add_argument("-m", "--markup", action="store_true",
                       help="Include comments with interval starts and ends")
    group.add_argument("-T", "--timestamp-raw", action="store_true",
                       help="Show raw timestamps in annotated information")
    group.add_argument("-c", "--count", action="store_true",
                       help="Just output a count of matched data points")
    return cmd

def cmd_extract_verify(self):
    if self.args.start is not None and self.args.end is not None:
        if self.args.start > self.args.end:
            self.parser.error("start is after end")

    if self.args.binary:
        if (self.args.bare or self.args.annotate or self.args.markup or
            self.args.timestamp_raw or self.args.count):
            self.parser.error("--binary cannot be combined with other options")

def cmd_extract(self):
    streams = self.client.stream_list(self.args.path)
    if len(streams) != 1:
        self.die("error getting stream info for path %s", self.args.path)
    layout = streams[0][1]

    if self.args.timestamp_raw:
```

```
60          time_string = nilmdb.utils.time.timestamp_to_string
61      else:
62          time_string = nilmdb.utils.time.timestamp_to_human
63
64      if self.args.annotate:
65          printf("# path: %s\n", self.args.path)
66          printf("# layout: %s\n", layout)
67          printf("# start: %s\n", time_string(self.args.start))
68          printf("# end: %s\n", time_string(self.args.end))
69
70      printed = False
71      if self.args.binary:
72          printer = sys.stdout.write
73      else:
74          printer = print
75      bare = self.args.bare
76      count = self.args.count
77      for dataline in self.client.stream_extract(self.args.path,
78                                                 self.args.start,
79                                                 self.args.end,
80                                                 self.args.count,
81                                                 self.args.markup,
82                                                 self.args.binary):
83          if bare and not count:
84              # Strip timestamp (first element).  Doesn't make sense
85              # if we are only returning a count.
86              dataline = ' '.join(dataline.split(' ')[1:])
87          printer(dataline)
88          printed = True
89      if not printed:
90          if self.args.annotate:
91              printf("# no data\n")
92          return 2
93
94      return 0
```

Listing B-22: `nilmdb/cmdline/help.py`: Handler for subcommand `help`: show detailed subcommand help.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/help.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   from nilmdb.utils.printf import *
2
3   import argparse
4   import sys
5
6   def setup(self, sub):
7       cmd = sub.add_parser("help", help="Show detailed help for a command",
8                            description="""
9                            Show help for a command. 'help command' is
10                           the same as 'command --help'.
11                           """)
12      cmd.set_defaults(handler = cmd_help)
13      cmd.set_defaults(no_test_connect = True)
14      cmd.add_argument("command", nargs="?",
15                       help="Command to get help about")
16      cmd.add_argument("rest", nargs=argparse.REMAINDER,
17                       help=argparse.SUPPRESS)
18      return cmd
19
20  def cmd_help(self):
```

```
21      if self.args.command in self.subcmd:
22          self.subcmd[self.args.command].print_help()
23      else:
24          self.parser.print_help()
25
26      return
```

Listing B-23: `nilmdb/cmdline/info.py`: Handler for subcommand `info`: show information about the server.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/info.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   import nilmdb.client
2   from nilmdb.utils.printf import *
3   from nilmdb.utils import human_size
4
5   from argparse import ArgumentDefaultsHelpFormatter as def_form
6
7   def setup(self, sub):
8       cmd = sub.add_parser("info", help="Server information",
9                            formatter_class = def_form,
10                           description="""
11                           List information about the server, like
12                           version.
13                           """)
14      cmd.set_defaults(handler = cmd_info)
15      return cmd
16
17  def cmd_info(self):
18      """Print info about the server"""
19      printf("Client version: %s\n", nilmdb.__version__)
20      printf("Server version: %s\n", self.client.version())
21      printf("Server URL: %s\n", self.client.geturl())
22      dbinfo = self.client.dbinfo()
23      printf("Server database path: %s\n", dbinfo["path"])
24      for (desc, field) in [("used by NilmDB", "size"),
25                            ("used by other", "other"),
26                            ("reserved", "reserved"),
27                            ("free", "free")]:
28          printf("Server disk space %s: %s\n", desc, human_size(dbinfo[field]))
```

Listing B-24: `nilmdb/cmdline/insert.py`: Handler for subcommand `insert`: insert data into a stream.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/insert.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   from nilmdb.utils.printf import *
2   import nilmdb.client
3   import nilmdb.utils.timestamper as timestamper
4   import nilmdb.utils.time
5
6   import sys
7
8   def setup(self, sub):
9       cmd = sub.add_parser("insert", help="Insert data",
10                           description="""
```

```
                        Insert data into a stream.
                        """)
    cmd.set_defaults(verify = cmd_insert_verify,
                     handler = cmd_insert)
    cmd.add_argument("-q", "--quiet", action='store_true',
                     help='suppress unnecessary messages')

    group = cmd.add_argument_group("Timestamping",
                                   description="""
                                   To add timestamps, specify the
                                   arguments --timestamp and --rate,
                                   and provide a starting time.
                                   """)

    group.add_argument("-t", "--timestamp", action="store_true",
                       help="Add timestamps to each line")
    group.add_argument("-r", "--rate", type=float,
                       help="Data rate, in Hz",
                       ).completer = self.complete.rate

    group = cmd.add_argument_group("Start time",
                                   description="""
                                   Start time may be manually
                                   specified with --start, or guessed
                                   from the filenames using
                                   --filename.  Set the TZ environment
                                   variable to change the default
                                   timezone.""")

    exc = group.add_mutually_exclusive_group()
    exc.add_argument("-s", "--start",
                     metavar="TIME", type=self.arg_time,
                     help="Starting timestamp (free-form)",
                     ).completer = self.complete.time
    exc.add_argument("-f", "--filename", action="store_true",
                     help="Use filename to determine start time")

    group = cmd.add_argument_group("End time",
                                   description="""
                                   End time for the overall stream.
                                   (required when not using --timestamp).
                                   Set the TZ environment
                                   variable to change the default
                                   timezone.""")
    group.add_argument("-e", "--end",
                       metavar="TIME", type=self.arg_time,
                       help="Ending timestamp (free-form)",
                       ).completer = self.complete.time

    group = cmd.add_argument_group("Required parameters")
    group.add_argument("path",
                       help="Path of stream, e.g. /foo/bar",
                       ).completer = self.complete.path
    group.add_argument("file", nargs = '?', default='-',
                       help="File to insert (default: - (stdin))")
    return cmd

def cmd_insert_verify(self):
    if self.args.timestamp:
        if not self.args.rate:
            self.die("error: --rate is needed, but was not specified")
        if not self.args.filename and self.args.start is None:
            self.die("error: need --start or --filename when adding timestamps")
    else:
        if self.args.start is None or self.args.end is None:
            self.die("error: when not adding timestamps, --start and "
                     "--end are required")
```

```
78
79  def cmd_insert(self):
80      # Find requested stream
81      streams = self.client.stream_list(self.args.path)
82      if len(streams) != 1:
83          self.die("error getting stream info for path %s", self.args.path)
84
85      arg = self.args
86
87      try:
88          filename = arg.file
89          if filename == '-':
90              infile = sys.stdin
91          else:
92              try:
93                  infile = open(filename, "rb")
94              except IOError:
95                  self.die("error opening input file %s", filename)
96
97          if arg.start is None:
98              try:
99                  arg.start = nilmdb.utils.time.parse_time(filename)
100             except ValueError:
101                 self.die("error extracting start time from filename '%s'",
102                          filename)
103
104         if arg.timestamp:
105             data = timestamper.TimestamperRate(infile, arg.start, arg.rate)
106         else:
107             data = iter(lambda: infile.read(1048576), '')
108
109         # Print info
110         if not arg.quiet:
111             printf(" Input file: %s\n", filename)
112             printf(" Start time: %s\n",
113                    nilmdb.utils.time.timestamp_to_human(arg.start))
114             if arg.end:
115                 printf("   End time: %s\n",
116                        nilmdb.utils.time.timestamp_to_human(arg.end))
117             if arg.timestamp:
118                 printf("Timestamper: %s\n", str(data))
119
120         # Insert the data
121         self.client.stream_insert(arg.path, data, arg.start, arg.end)
122
123     except nilmdb.client.Error as e:
124         # TODO: It would be nice to be able to offer better errors
125         # here, particularly in the case of overlap, which just shows
126         # ugly bracketed ranges of 16-digit numbers and a mangled URL.
127         # Need to consider adding something like e.prettyprint()
128         # that is smarter about the contents of the error.
129         self.die("error inserting data: %s", str(e))
130
131     return
```

Listing B-25: `nilmdb/cmdline/intervals.py`: Handler for subcommand `intervals`: list intervals in a stream.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/cmdline/intervals.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
from nilmdb.utils.printf import *
import nilmdb.utils.time
from nilmdb.utils.interval import Interval

import fnmatch
import argparse
from argparse import ArgumentDefaultsHelpFormatter as def_form

def setup(self, sub):
    cmd = sub.add_parser("intervals", help="List intervals",
                         formatter_class = def_form,
                         description="""
                         List intervals in a stream, similar to
                         'list --detail path'.

                         If '--diff diffpath' is provided, only
                         interval ranges that are present in 'path'
                         and not present in 'diffpath' are printed.
                         """)
    cmd.set_defaults(verify = cmd_intervals_verify,
                     handler = cmd_intervals)

    group = cmd.add_argument_group("Stream selection")
    group.add_argument("path", metavar="PATH",
                       help="List intervals for this path",
                       ).completer = self.complete.path
    group.add_argument("-d", "--diff", metavar="PATH",
                       help="Subtract intervals from this path",
                       ).completer = self.complete.path

    group = cmd.add_argument_group("Interval details")
    group.add_argument("-s", "--start",
                       metavar="TIME", type=self.arg_time,
                       help="Starting timestamp for intervals "
                       "(free-form, inclusive)",
                       ).completer = self.complete.time
    group.add_argument("-e", "--end",
                       metavar="TIME", type=self.arg_time,
                       help="Ending timestamp for intervals "
                       "(free-form, noninclusive)",
                       ).completer = self.complete.time

    group = cmd.add_argument_group("Misc options")
    group.add_argument("-T", "--timestamp-raw", action="store_true",
                       help="Show raw timestamps when printing times")
    group.add_argument("-o", "--optimize", action="store_true",
                       help="Optimize (merge adjacent) intervals")

    return cmd

def cmd_intervals_verify(self):
    if self.args.start is not None and self.args.end is not None:
        if self.args.start >= self.args.end:
            self.parser.error("start must precede end")

def cmd_intervals(self):
    """List intervals in a stream"""
    if self.args.timestamp_raw:
        time_string = nilmdb.utils.time.timestamp_to_string
```

```
60      else:
61          time_string = nilmdb.utils.time.timestamp_to_human
62
63      try:
64          intervals = ( Interval(start, end) for (start, end) in
65                        self.client.stream_intervals(self.args.path,
66                                                     self.args.start,
67                                                     self.args.end,
68                                                     self.args.diff) )
69          if self.args.optimize:
70              intervals = nilmdb.utils.interval.optimize(intervals)
71          for i in intervals:
72              printf("[ %s -> %s ]\n", time_string(i.start), time_string(i.end))
73
74      except nilmdb.client.ClientError as e:
75          self.die("error listing intervals: %s", str(e))
```

Listing B-26: `nilmdb/cmdline/list.py`: Handler for subcommand `list`: list streams and stream info.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/list.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   from nilmdb.utils.printf import *
2   import nilmdb.utils.time
3
4   import fnmatch
5   import argparse
6   from argparse import ArgumentDefaultsHelpFormatter as def_form
7
8   def setup(self, sub):
9       cmd = sub.add_parser("list", help="List streams",
10                           formatter_class = def_form,
11                           description="""
12                           List streams available in the database,
13                           optionally filtering by path.  Wildcards
14                           are accepted; non-matching paths or wildcards
15                           are ignored.
16                           """)
17      cmd.set_defaults(verify = cmd_list_verify,
18                       handler = cmd_list)
19
20      group = cmd.add_argument_group("Stream filtering")
21      group.add_argument("path", metavar="PATH", default=["*"], nargs='*',
22                         ).completer = self.complete.path
23
24      group = cmd.add_argument_group("Interval info")
25      group.add_argument("-E", "--ext", action="store_true",
26                         help="Show extended stream info, like interval "
27                         "extents and row count")
28
29      group = cmd.add_argument_group("Interval details")
30      group.add_argument("-d", "--detail", action="store_true",
31                         help="Show available data time intervals")
32      group.add_argument("-s", "--start",
33                         metavar="TIME", type=self.arg_time,
34                         help="Starting timestamp for intervals "
35                         "(free-form, inclusive)",
36                         ).completer = self.complete.time
37      group.add_argument("-e", "--end",
38                         metavar="TIME", type=self.arg_time,
39                         help="Ending timestamp for intervals "
```

```
                        "(free-form, noninclusive)",
                        ).completer = self.complete.time

    group = cmd.add_argument_group("Misc options")
    group.add_argument("-T", "--timestamp-raw", action="store_true",
                        help="Show raw timestamps when printing times")
    group.add_argument("-l", "--layout", action="store_true",
                        help="Show layout type next to path name")
    group.add_argument("-n", "--no-decim", action="store_true",
                        help="Skip paths containing \"~decim-\"")

    return cmd

def cmd_list_verify(self):
    if self.args.start is not None and self.args.end is not None:
        if self.args.start >= self.args.end:
            self.parser.error("start must precede end")

    if self.args.start is not None or self.args.end is not None:
        if not self.args.detail:
            self.parser.error("--start and --end only make sense with --detail")

def cmd_list(self):
    """List available streams"""
    streams = self.client.stream_list(extended = True)

    if self.args.timestamp_raw:
        time_string = nilmdb.utils.time.timestamp_to_string
    else:
        time_string = nilmdb.utils.time.timestamp_to_human

    for argpath in self.args.path:
        for stream in streams:
            (path, layout, int_min, int_max, rows, time) = stream[:6]
            if not fnmatch.fnmatch(path, argpath):
                continue
            if self.args.no_decim and "~decim-" in path:
                continue

            if self.args.layout:
                printf("%s %s\n", path, layout)
            else:
                printf("%s\n", path)

            if self.args.ext:
                if int_min is None or int_max is None:
                    printf("  interval extents: (no data)\n")
                else:
                    printf("  interval extents: %s -> %s\n",
                           time_string(int_min), time_string(int_max))
                printf("        total data: %d rows, %.6f seconds\n",
                       rows or 0,
                       nilmdb.utils.time.timestamp_to_seconds(time or 0))

            if self.args.detail:
                printed = False
                for (start, end) in self.client.stream_intervals(
                    path, self.args.start, self.args.end):
                    printf("  [ %s -> %s ]\n",
                           time_string(start), time_string(end))
                    printed = True
                if not printed:
                    printf("  (no intervals)\n")
```

Listing B-27: `nilmdb/cmdline/metadata.py`: Handler for subcommand `metadata`: manage stream metadata.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/cmdline/metadata.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
from nilmdb.utils.printf import *
import nilmdb
import nilmdb.client

def setup(self, sub):
    cmd = sub.add_parser("metadata", help="Get or set stream metadata",
                         description="""
                         Get or set key=value metadata associated with
                         a stream.
                         """,
                         usage="%(prog)s path [-g [key ...] | "
                         "-s key=value [...] | -u key=value [...]] | "
                         "-d [key ...]")
    cmd.set_defaults(handler = cmd_metadata)

    group = cmd.add_argument_group("Required arguments")
    group.add_argument("path",
                       help="Path of stream, e.g. /foo/bar",
                       ).completer = self.complete.path

    group = cmd.add_argument_group("Actions")
    exc = group.add_mutually_exclusive_group()
    exc.add_argument("-g", "--get", nargs="*", metavar="key",
                     help="Get metadata for specified keys (default all)",
                     ).completer = self.complete.meta_key
    exc.add_argument("-s", "--set", nargs="+", metavar="key=value",
                     help="Replace all metadata with provided "
                     "key=value pairs",
                     ).completer = self.complete.meta_keyval
    exc.add_argument("-u", "--update", nargs="+", metavar="key=value",
                     help="Update metadata using provided "
                     "key=value pairs",
                     ).completer = self.complete.meta_keyval
    exc.add_argument("-d", "--delete", nargs="*", metavar="key",
                     help="Delete metadata for specified keys (default all)",
                     ).completer = self.complete.meta_key
    return cmd

def cmd_metadata(self):
    """Manipulate metadata"""
    if self.args.set is not None or self.args.update is not None:
        # Either set, or update
        if self.args.set is not None:
            keyvals = map(nilmdb.utils.unicode.decode, self.args.set)
            handler = self.client.stream_set_metadata
        else:
            keyvals = map(nilmdb.utils.unicode.decode, self.args.update)
            handler = self.client.stream_update_metadata

        # Extract key=value pairs
        data = {}
        for keyval in keyvals:
            kv = keyval.split('=')
            if len(kv) != 2 or kv[0] == "":
                self.die("error parsing key=value argument '%s'", keyval)
            data[kv[0]] = kv[1]

        # Make the call
        try:
```

```
60              handler(self.args.path, data)
61          except nilmdb.client.ClientError as e:
62              self.die("error setting/updating metadata: %s", str(e))
63      elif self.args.delete is not None:
64          # Delete (by setting values to empty strings)
65          keys = None
66          if self.args.delete:
67              keys = map(nilmdb.utils.unicode.decode, self.args.delete)
68          try:
69              data = self.client.stream_get_metadata(self.args.path, keys)
70              for key in data:
71                  data[key] = ""
72              self.client.stream_update_metadata(self.args.path, data)
73          except nilmdb.client.ClientError as e:
74              self.die("error deleting metadata: %s", str(e))
75      else:
76          # Get (or unspecified)
77          keys = None
78          if self.args.get:
79              keys = map(nilmdb.utils.unicode.decode, self.args.get)
80          try:
81              data = self.client.stream_get_metadata(self.args.path, keys)
82          except nilmdb.client.ClientError as e:
83              self.die("error getting metadata: %s", str(e))
84          for key, value in sorted(data.items()):
85              # Print nonexistant keys as having empty value
86              if value is None:
87                  value = ""
88              printf("%s=%s\n",
89                     nilmdb.utils.unicode.encode(key),
90                     nilmdb.utils.unicode.encode(value))
```

Listing B-28: `nilmdb/cmdline/remove.py`: Handler for subcommand `remove`: remove data from streams.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/cmdline/remove.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   from nilmdb.utils.printf import *
2   import nilmdb.client
3   import fnmatch
4
5   def setup(self, sub):
6       cmd = sub.add_parser("remove", help="Remove data",
7                            description="""
8                            Remove all data from a specified time range within a
9                            stream.  If multiple streams or wildcards are provided,
10                           the same time range is removed from all streams.
11                           """)
12      cmd.set_defaults(handler = cmd_remove)
13
14      group = cmd.add_argument_group("Data selection")
15      group.add_argument("path", nargs='+',
16                         help="Path of stream, e.g. /foo/bar/*",
17                         ).completer = self.complete.path
18      group.add_argument("-s", "--start", required=True,
19                         metavar="TIME", type=self.arg_time,
20                         help="Starting timestamp (free-form, inclusive)",
21                         ).completer = self.complete.time
22      group.add_argument("-e", "--end", required=True,
23                         metavar="TIME", type=self.arg_time,
24                         help="Ending timestamp (free-form, noninclusive)",
```

```
                        ).completer = self.complete.time

    group = cmd.add_argument_group("Output format")
    group.add_argument("-q", "--quiet", action="store_true",
                       help="Don't display names when removing "
                       "from multiple paths")
    group.add_argument("-c", "--count", action="store_true",
                       help="Output number of data points removed")
    return cmd

def cmd_remove(self):
    streams = [ s[0] for s in self.client.stream_list() ]
    paths = []
    for path in self.args.path:
        new = fnmatch.filter(streams, path)
        if not new:
            self.die("error: no stream matched path: %s", path)
        paths.extend(new)

    try:
        for path in paths:
            if not self.args.quiet and len(paths) > 1:
                printf("Removing from %s\n", path)
            count = self.client.stream_remove(path,
                                              self.args.start, self.args.end)
            if self.args.count:
                printf("%d\n", count);
    except nilmdb.client.ClientError as e:
        self.die("error removing data: %s", str(e))

    return 0
```

Listing B-29: `nilmdb/cmdline/rename.py`: Handler for subcommand `rename`: rename streams.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/cmdline/rename.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
from nilmdb.utils.printf import *
import nilmdb.client

from argparse import ArgumentDefaultsHelpFormatter as def_form

def setup(self, sub):
    cmd = sub.add_parser("rename", help="Rename a stream",
                         formatter_class = def_form,
                         description="""
                         Rename a stream.

                         Only the stream's path is renamed; no
                         metadata is changed.
                         """)
    cmd.set_defaults(handler = cmd_rename)
    group = cmd.add_argument_group("Required arguments")
    group.add_argument("oldpath",
                       help="Old path, e.g. /foo/old",
                       ).completer = self.complete.path
    group.add_argument("newpath",
                       help="New path, e.g. /foo/bar/new",
                       ).completer = self.complete.path

    return cmd
```

```
25
26  def cmd_rename(self):
27      """Rename a stream"""
28      try:
29          self.client.stream_rename(self.args.oldpath, self.args.newpath)
30      except nilmdb.client.ClientError as e:
31          self.die("error renaming stream: %s", str(e))
```

B.5 Shared Utilities

Listing B-30: `nilmdb/utils/__init__.py`: Shared utility module initialization.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/__init__.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   """NilmDB utilities"""
2
3   from __future__ import absolute_import
4   from nilmdb.utils.timer import Timer
5   from nilmdb.utils.serializer import serializer_proxy
6   from nilmdb.utils.lrucache import lru_cache
7   from nilmdb.utils.diskusage import du, human_size
8   from nilmdb.utils.mustclose import must_close
9   from nilmdb.utils import atomic
10  import nilmdb.utils.threadsafety
11  import nilmdb.utils.fallocate
12  import nilmdb.utils.time
13  import nilmdb.utils.iterator
14  import nilmdb.utils.interval
15  import nilmdb.utils.lock
16  import nilmdb.utils.sort
17  import nilmdb.utils.unicode
```

Listing B-31: `nilmdb/utils/atomic.py`: Atomic file writing helper. Replaces one file with another in a manner that is likely to leave a valid file in the event of unclean system shutdown.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/atomic.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   # Atomic file writing helper.
2
3   import os
4
5   def replace_file(filename, content):
6       """Attempt to atomically and durably replace the filename with the
7       given contents.  This is intended to be 'pretty good on most
8       OSes', but not necessarily bulletproof."""
9
10      newfilename = filename + ".new"
11
12      # Write to new file, flush it
13      with open(newfilename, "wb") as f:
14          f.write(content)
15          f.flush()
```

```
            os.fsync(f.fileno())

        # Move new file over old one
        try:
            os.rename(newfilename, filename)
        except OSError: # pragma: no cover
            # Some OSes might not support renaming over an existing file.
            # This is definitely NOT atomic!
            os.remove(filename)
            os.rename(newfilename, filename)
```

Listing B-32: `nilmdb/utils/diskusage.py`: Disk usage measurement.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/diskusage.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
import os
import errno
from math import log

def human_size(num):
    """Human friendly file size"""
    unit_list = zip(['bytes', 'kiB', 'MiB', 'GiB', 'TiB'], [0, 0, 1, 2, 2])
    if num > 1:
        exponent = min(int(log(num, 1024)), len(unit_list) - 1)
        quotient = float(num) / 1024**exponent
        unit, num_decimals = unit_list[exponent]
        format_string = '{:.%sf} {}' % (num_decimals)
        return format_string.format(quotient, unit)
    if num == 0: # pragma: no cover
        return '0 bytes'
    if num == 1: # pragma: no cover
        return '1 byte'

def du(path):
    """Like du -sb, returns total size of path in bytes.  Ignore
    errors that might occur if we encounter broken symlinks or
    files in the process of being removed."""
    try:
        st = os.stat(path)
        size = st.st_blocks * 512
        if os.path.isdir(path):
            for thisfile in os.listdir(path):
                filepath = os.path.join(path, thisfile)
                size += du(filepath)
        return size
    except OSError as e: # pragma: no cover
        if e.errno != errno.ENOENT:
            raise
        return 0
```

Listing B-33: `nilmdb/utils/fallocate.py`: Punch holes in files using the `fallocate` system call. This creates sparse files on modern filesystems that support it, which permits disk space recovery when rows are removed from the middle of a particular bulkdata storage file.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/utils/fallocate.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# Implementation of hole punching via fallocate, if the OS
# and filesystem support it.

try:
    import os
    import ctypes
    import ctypes.util

    def make_fallocate():
        libc_name = ctypes.util.find_library('c')
        libc = ctypes.CDLL(libc_name, use_errno=True)

        _fallocate = libc.fallocate
        _fallocate.restype = ctypes.c_int
        _fallocate.argtypes = [ ctypes.c_int, ctypes.c_int,
                                ctypes.c_int64, ctypes.c_int64 ]

        del libc
        del libc_name

        def fallocate(fd, mode, offset, len_):
            res = _fallocate(fd, mode, offset, len_)
            if res != 0: # pragma: no cover
                errno = ctypes.get_errno()
                raise IOError(errno, os.strerror(errno))
        return fallocate

    fallocate = make_fallocate()
    del make_fallocate
except Exception: # pragma: no cover
    fallocate = None

FALLOC_FL_KEEP_SIZE = 0x01
FALLOC_FL_PUNCH_HOLE = 0x02

def punch_hole(filename, offset, length, ignore_errors = True):
    """Punch a hole in the file.  This isn't well supported, so errors
    are ignored by default."""
    try:
        if fallocate is None: # pragma: no cover
            raise IOError("fallocate not available")
        with open(filename, "r+") as f:
            fallocate(f.fileno(),
                      FALLOC_FL_KEEP_SIZE | FALLOC_FL_PUNCH_HOLE,
                      offset, length)
    except IOError: # pragma: no cover
        if ignore_errors:
            return
        raise
```

Listing B-34: `nilmdb/utils/interval.py`: Interval class and interval set operations.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/interval.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
"""Interval.  Like nilmdb.server.interval, but re-implemented here
in plain Python so clients have easier access to it, and with a few
helper functions.

Intervals are half-open, ie. they include data points with timestamps
[start, end)
"""

import nilmdb.utils.time
import nilmdb.utils.iterator

class IntervalError(Exception):
    """Error due to interval overlap, etc"""
    pass

# Interval
class Interval:
    """Represents an interval of time."""

    def __init__(self, start, end):
        """
        'start' and 'end' are arbitrary numbers that represent time
        """
        if start >= end:
            # Explicitly disallow zero-width intervals (since they're half-open)
            raise IntervalError("start %s must precede end %s" % (start, end))
        self.start = start
        self.end = end

    def __repr__(self):
        s = repr(self.start) + ", " + repr(self.end)
        return self.__class__.__name__ + "(" + s + ")"

    def __str__(self):
        return ("[" + nilmdb.utils.time.timestamp_to_string(self.start) +
                " -> " + nilmdb.utils.time.timestamp_to_string(self.end) + ")")

    def human_string(self):
        return ("[ " + nilmdb.utils.time.timestamp_to_human(self.start) +
                " -> " + nilmdb.utils.time.timestamp_to_human(self.end) + " ]")

    def __cmp__(self, other):
        """Compare two intervals.  If non-equal, order by start then end"""
        return cmp(self.start, other.start) or cmp(self.end, other.end)

    def intersects(self, other):
        """Return True if two Interval objects intersect"""
        if not isinstance(other, Interval):
            raise TypeError("need an Interval")
        if self.end <= other.start or self.start >= other.end:
            return False
        return True

    def subset(self, start, end):
        """Return a new Interval that is a subset of this one"""
        # A subclass that tracks additional data might override this.
        if start < self.start or end > self.end:
            raise IntervalError("not a subset")
        return Interval(start, end)
```

```
60
61  def _interval_math_helper(a, b, op, subset = True):
62      """Helper for set_difference, intersection functions,
63      to compute interval subsets based on a math operator on ranges
64      present in A and B.  Subsets are computed from A, or new intervals
65      are generated if subset = False."""
66      # Iterate through all starts and ends in sorted order.  Add a
67      # tag to the iterator so that we can figure out which one they
68      # were, after sorting.
69      def decorate(it, key_start, key_end):
70          for i in it:
71              yield i.start, key_start, i
72              yield i.end, key_end, i
73      a_iter = decorate(iter(a), 0, 2)
74      b_iter = decorate(iter(b), 1, 3)
75
76      # Now iterate over the timestamps of each start and end.
77      # At each point, evaluate which type of end it is, to determine
78      # how to build up the output intervals.
79      a_interval = None
80      in_a = False
81      in_b = False
82      out_start = None
83      for (ts, k, i) in nilmdb.utils.iterator.imerge(a_iter, b_iter):
84          if k == 0:
85              a_interval = i
86              in_a = True
87          elif k == 1:
88              in_b = True
89          elif k == 2:
90              in_a = False
91          elif k == 3:
92              in_b = False
93          include = op(in_a, in_b)
94          if include and out_start is None:
95              out_start = ts
96          elif not include:
97              if out_start is not None and out_start != ts:
98                  if subset:
99                      yield a_interval.subset(out_start, ts)
100                 else:
101                     yield Interval(out_start, ts)
102             out_start = None
103
104 def set_difference(a, b):
105     """
106     Compute the difference (a \\ b) between the intervals in 'a' and
107     the intervals in 'b'; i.e., the ranges that are present in 'self'
108     but not 'other'.
109
110     'a' and 'b' must both be iterables.
111
112     Returns a generator that yields each interval in turn.
113     Output intervals are built as subsets of the intervals in the
114     first argument (a).
115     """
116     return _interval_math_helper(a, b, (lambda a, b: a and not b))
117
118 def intersection(a, b):
119     """
120     Compute the intersection between the intervals in 'a' and the
121     intervals in 'b'; i.e., the ranges that are present in both 'a'
122     and 'b'.
123
124     'a' and 'b' must both be iterables.
125
126     Returns a generator that yields each interval in turn.
```

```
    Output intervals are built as subsets of the intervals in the
    first argument (a).
    """
    return _interval_math_helper(a, b, (lambda a, b: a and b))

def optimize(it):
    """
    Given an iterable 'it' with intervals, optimize them by joining
    together intervals that are adjacent in time, and return a generator
    that yields the new intervals.
    """
    saved_int = None
    for interval in it:
        if saved_int is not None:
            if saved_int.end == interval.start:
                interval.start = saved_int.start
            else:
                yield saved_int
        saved_int = interval
    if saved_int is not None:
        yield saved_int
```

Listing B-35: `nilmdb/utils/iterator.py`: Miscellaneous iterator tools.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/iterator.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
# Misc iterator tools

# Iterator merging, based on http://code.activestate.com/recipes/491285/
import heapq
def imerge(*iterables):
    '''Merge multiple sorted inputs into a single sorted output.

    Equivalent to:  sorted(itertools.chain(*iterables))

    >>> list(imerge([1,3,5,7], [0,2,4,8], [5,10,15,20], [], [25]))
    [0, 1, 2, 3, 4, 5, 5, 7, 8, 10, 15, 20, 25]

    '''
    heappop, siftup, _Stop = heapq.heappop, heapq._siftup, StopIteration

    h = []
    h_append = h.append
    for it in map(iter, iterables):
        try:
            next = it.next
            h_append([next(), next])
        except _Stop:
            pass
    heapq.heapify(h)

    while 1:
        try:
            while 1:
                v, next = s = h[0]      # raises IndexError when h is empty
                yield v
                s[0] = next()           # raises StopIteration when exhausted
                siftup(h, 0)            # restore heap condition
        except _Stop:
            heappop(h)                  # remove empty iterator
        except IndexError:
            return
```

Listing B-36: `nilmdb/utils/lock.py`: Mandatory file locking. A dummy function is used on operating systems that do not support `fcntl.flock`.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/utils/lock.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# File locking

import warnings

try:
    import fcntl
    import errno

    def exclusive_lock(f):
        """Acquire an exclusive lock.  Returns True on successful
        lock, or False on error."""
        try:
            fcntl.flock(f.fileno(), fcntl.LOCK_EX | fcntl.LOCK_NB)
        except IOError as e:
            if e.errno in (errno.EACCES, errno.EAGAIN):
                return False
            else: # pragma: no cover
                raise
        return True

    def exclusive_unlock(f):
        """Release an exclusive lock."""
        fcntl.flock(f.fileno(), fcntl.LOCK_UN)

except ImportError: # pragma: no cover
    def exclusive_lock(f):
        """Dummy lock function -- does not lock!"""
        warnings.warn("Pretending to lock " + str(f))
        return True

    def exclusive_unlock(f):
        """Release an exclusive lock."""
        return
```

Listing B-37: `nilmdb/utils/lrucache.py`: Memoization decorator for function return values. This decorator maintains a cache of return values from a function. Subsequent calls with the same parameters will return the cached values. Cache entries are evicted in a LRU fashion or can be manually managed.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/utils/lrucache.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# Memoize a function's return value with a least-recently-used cache
# Based on:
#   http://code.activestate.com/recipes/498245-lru-and-lfu-cache-decorators/
# with added 'destructor' functionality.

```

```
import collections
import decorator

def lru_cache(size = 10, onremove = None, keys = slice(None)):
    """Least-recently-used cache decorator.

    @lru_cache(size = 10, onevict = None)
    def f(...):
        pass

    Given a function and arguments, memoize its return value.  Up to
    'size' elements are cached.  'keys' is a slice object that
    represents which arguments are used as the cache key.

    When evicting a value from the cache, call the function
    'onremove' with the value that's being evicted.

    Call f.cache_remove(...) to evict the cache entry with the given
    arguments.  Call f.cache_remove_all() to evict all entries.
    f.cache_hits and f.cache_misses give statistics.
    """

    def decorate(func):
        cache = collections.OrderedDict()   # order: least- to most-recent

        def evict(value):
            if onremove:
                onremove(value)

        def wrapper(orig, *args, **kwargs):
            if kwargs:
                raise NotImplementedError("kwargs not supported")
            key = args[keys]
            try:
                value = cache.pop(key)
                orig.cache_hits += 1
            except KeyError:
                value = orig(*args)
                orig.cache_misses += 1
                if len(cache) >= size:
                    evict(cache.popitem(0)[1])  # evict LRU cache entry
                cache[key] = value              # (re-)insert this key at end
            return value

        def cache_remove(*args):
            """Remove the described key from this cache, if present."""
            key = args
            if key in cache:
                evict(cache.pop(key))
            else:
                if len(cache) > 0 and len(args) != len(cache.iterkeys().next()):
                    raise KeyError("trying to remove from LRU cache, but "
                                   "number of arguments doesn't match the "
                                   "cache key length")

        def cache_remove_all():
            for key in cache:
                evict(cache.pop(key))

        def cache_info():
            return (func.cache_hits, func.cache_misses)

        new = decorator.decorator(wrapper, func)
        func.cache_hits = 0
        func.cache_misses = 0
        new.cache_info = cache_info
        new.cache_remove = cache_remove
```

```
73          new.cache_remove_all = cache_remove_all
74          return new
75
76      return decorate
```

Listing B-38: `nilmdb/utils/mustclose.py`: Class decorator that ensures `close()` functions are called. This is used to ensure correctness in functions that maintain internal state and perform bookkeeping on `close()`.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/mustclose.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   from nilmdb.utils.printf import *
2   import sys
3   import inspect
4   import decorator
5
6   def must_close(errorfile = sys.stderr, wrap_verify = False):
7       """Class decorator that warns on 'errorfile' at deletion time if
8       the class's close() member wasn't called.
9
10      If 'wrap_verify' is True, every class method is wrapped with a
11      verifier that will raise AssertionError if the .close() method has
12      already been called."""
13      def class_decorator(cls):
14
15          def wrap_class_method(wrapper):
16              try:
17                  orig = getattr(cls, wrapper.__name__).im_func
18              except Exception:
19                  orig = lambda x: None
20              setattr(cls, wrapper.__name__, decorator.decorator(wrapper, orig))
21
22          @wrap_class_method
23          def __init__(orig, self, *args, **kwargs):
24              ret = orig(self, *args, **kwargs)
25              self.__dict__["_must_close"] = True
26              self.__dict__["_must_close_initialized"] = True
27              return ret
28
29          @wrap_class_method
30          def __del__(orig, self, *args, **kwargs):
31              if "_must_close" in self.__dict__:
32                  fprintf(errorfile, "error: %s.close() wasn't called!\n",
33                          self.__class__.__name__)
34              return orig(self, *args, **kwargs)
35
36          @wrap_class_method
37          def close(orig, self, *args, **kwargs):
38              if "_must_close" in self.__dict__:
39                  del self._must_close
40              return orig(self, *args, **kwargs)
41
42          # Optionally wrap all other functions
43          def verifier(orig, self, *args, **kwargs):
44              if ("_must_close" not in self.__dict__ and
45                  "_must_close_initialized" in self.__dict__):
46                  raise AssertionError("called " + str(orig) + " after close")
47              return orig(self, *args, **kwargs)
48          if wrap_verify:
```

```
            for (name, method) in inspect.getmembers(cls, inspect.ismethod):
                # Skip class methods
                if method.__self__ is not None:
                    continue
                # Skip some methods
                if name in [ "__del__", "__init__" ]:
                    continue
                # Set up wrapper
                setattr(cls, name, decorator.decorator(verifier,
                                                       method.im_func))

        return cls
    return class_decorator
```

Listing B-39: `nilmdb/utils/printf.py`: Implementation of C-like `printf` family of functions.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/printf.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
"""printf, fprintf, sprintf"""

from __future__ import print_function
def printf(_str, *args):
    print(_str % args, end='')
def fprintf(_file, _str, *args):
    print(_str % args, end='', file=_file)
def sprintf(_str, *args):
    return (_str % args)
```

Listing B-40: `nilmdb/utils/serializer.py`: Serialization wrapper for method calls from multiple threads. This provides a proxy object that wraps all calls to another object in a thread-safe manner. Method calls on the proxy object are queued in a first-in, first-out manner, and are executed from a single dedicated thread. Return values and exceptions are propagated back to the caller when ready.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/serializer.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
import Queue
import threading
import sys
import decorator
import inspect
import types
import functools

# This file provides a class that will wrap an object and serialize
# all calls to its methods.  All calls to that object will be queued
# and executed from a single thread, regardless of which thread makes
# the call.

# Based partially on http://stackoverflow.com/questions/2642515/

class SerializerThread(threading.Thread):
```

```
    """Thread that retrieves call information from the queue, makes the
    call, and returns the results."""
    def __init__(self, classname, call_queue):
        threading.Thread.__init__(self)
        self.name = "Serializer-" + classname + "-" + self.name
        self.call_queue = call_queue

    def run(self):
        while True:
            result_queue, func, args, kwargs = self.call_queue.get()
            # Terminate if result_queue is None
            if result_queue is None:
                return
            exception = None
            result = None
            try:
                result = func(*args, **kwargs) # wrapped
            except:
                exception = sys.exc_info()
            # Ensure we delete these before returning a result, so
            # we don't unncessarily hold onto a reference while
            # we're waiting for the next call.
            del func, args, kwargs
            result_queue.put((exception, result))
            del exception, result

def serializer_proxy(obj_or_type):
    """Wrap the given object or type in a SerializerObjectProxy.

    Returns a SerializerObjectProxy object that proxies all method
    calls to the object, as well as attribute retrievals.

    The proxied requests, including instantiation, are performed in a
    single thread and serialized between caller threads.
    """
    class SerializerCallProxy(object):
        def __init__(self, call_queue, func, objectproxy):
            self.call_queue = call_queue
            self.func = func
            # Need to hold a reference to object proxy so it doesn't
            # go away (and kill the thread) until after get called.
            self.objectproxy = objectproxy
        def __call__(self, *args, **kwargs):
            result_queue = Queue.Queue()
            self.call_queue.put((result_queue, self.func, args, kwargs))
            ( exc_info, result ) = result_queue.get()
            if exc_info is None:
                return result
            else:
                raise exc_info[0], exc_info[1], exc_info[2]

    class SerializerObjectProxy(object):
        def __init__(self, obj_or_type, *args, **kwargs):
            self.__object = obj_or_type
            try:
                if type(obj_or_type) in (types.TypeType, types.ClassType):
                    classname = obj_or_type.__name__
                else:
                    classname = obj_or_type.__class__.__name__
            except AttributeError: # pragma: no cover
                classname = "???"
            self.__call_queue = Queue.Queue()
            self.__thread = SerializerThread(classname, self.__call_queue)
            self.__thread.daemon = True
            self.__thread.start()
            self._thread_safe = True

```

```
84          def __getattr__(self, key):
85              if key.startswith("_SerializerObjectProxy__"): # pragma: no cover
86                  raise AttributeError
87              attr = getattr(self.__object, key)
88              if not callable(attr):
89                  getter = SerializerCallProxy(self.__call_queue, getattr, self)
90                  return getter(self.__object, key)
91              r = SerializerCallProxy(self.__call_queue, attr, self)
92              return r
93
94          # For an interable object, on __iter__(), save the object's
95          # iterator and return this proxy.  On next(), call the object's
96          # iterator through this proxy.
97          def __iter__(self):
98              attr = getattr(self.__object, "__iter__")
99              self.__iter = SerializerCallProxy(self.__call_queue, attr, self)()
100             return self
101         def next(self):
102             return SerializerCallProxy(self.__call_queue,
103                                        self.__iter.next, self)()
104
105         def __getitem__(self, key):
106             return self.__getattr__("__getitem__")(key)
107
108         def __call__(self, *args, **kwargs):
109             """Call this to instantiate the type, if a type was passed
110             to serializer_proxy.  Otherwise, pass the call through."""
111             ret = SerializerCallProxy(self.__call_queue,
112                                       self.__object, self)(*args, **kwargs)
113             if type(self.__object) in (types.TypeType, types.ClassType):
114                 # Instantiation
115                 self.__object = ret
116                 return self
117             return ret
118
119         def __del__(self):
120             self.__call_queue.put((None, None, None, None))
121             self.__thread.join()
122
123     return SerializerObjectProxy(obj_or_type)
```

Listing B-41: `nilmdb/utils/sort.py`: Human-friendly string sorting routines.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/sort.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   import re
2
3   def sort_human(items, key = None):
4       """Human-friendly sort (/stream/2 before /stream/10)"""
5       def to_num(val):
6           try:
7               return int(val)
8           except Exception:
9               return val
10
11      def human_key(text):
12          if key:
13              text = key(text)
14          # Break into character and numeric chunks.
15          chunks = re.split(r'([0-9]+)', text)
16          return [ to_num(c) for c in chunks ]
17
```

```
18      return sorted(items, key = human_key)
```

Listing B-42: `nilmdb/utils/threadsafety.py`: Wrapper to verify thread safety for method calls. This is used to ensure correctness in objects and types that are known not to be thread-safe. It is typically used in conjunction with the `serializer_proxy`.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `nilmdb/utils/threadsafety.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1   from nilmdb.utils.printf import *
2   import threading
3   import warnings
4   import types
5
6   def verify_proxy(obj_or_type, exception = False, check_thread = True,
7                    check_concurrent = True):
8       """Wrap the given object or type in a VerifyObjectProxy.
9
10      Returns a VerifyObjectProxy that proxies all method calls to the
11      given object, as well as attribute retrievals.
12
13      When calling methods, the following checks are performed.  If
14      exception is True, an exception is raised.  Otherwise, a warning
15      is printed.
16
17      check_thread = True     # Warn/fail if two different threads call methods.
18      check_concurrent = True # Warn/fail if two functions are concurrently
19                              # run through this proxy
20      """
21      class Namespace(object):
22          pass
23      class VerifyCallProxy(object):
24          def __init__(self, func, parent_namespace):
25              self.func = func
26              self.parent_namespace = parent_namespace
27
28          def __call__(self, *args, **kwargs):
29              p = self.parent_namespace
30              this = threading.current_thread()
31              try:
32                  callee = self.func.__name__
33              except AttributeError:
34                  callee = "???"
35
36              if p.thread is None:
37                  p.thread = this
38                  p.thread_callee = callee
39
40              if check_thread and p.thread != this:
41                  err = sprintf("unsafe threading: %s called %s.%s,"
42                                " but %s called %s.%s",
43                                p.thread.name, p.classname, p.thread_callee,
44                                this.name, p.classname, callee)
45                  if exception:
46                      raise AssertionError(err)
47                  else: # pragma: no cover
48                      warnings.warn(err)
49
50              need_concur_unlock = False
51              if check_concurrent:
```

```
                if p.concur_lock.acquire(False) == False:
                    err = sprintf("unsafe concurrency: %s called %s.%s "
                                  "while %s is still in %s.%s",
                                  this.name, p.classname, callee,
                                  p.concur_tname, p.classname, p.concur_callee)
                    if exception:
                        raise AssertionError(err)
                    else: # pragma: no cover
                        warnings.warn(err)
                else:
                    p.concur_tname = this.name
                    p.concur_callee = callee
                    need_concur_unlock = True

            try:
                ret = self.func(*args, **kwargs)
            finally:
                if need_concur_unlock:
                    p.concur_lock.release()
            return ret

    class VerifyObjectProxy(object):
        def __init__(self, obj_or_type, *args, **kwargs):
            p = Namespace()
            self.__ns = p
            p.thread = None
            p.thread_callee = None
            p.concur_lock = threading.Lock()
            p.concur_tname = None
            p.concur_callee = None
            self.__obj = obj_or_type
            try:
                if type(obj_or_type) in (types.TypeType, types.ClassType):
                    p.classname = self.__obj.__name__
                else:
                    p.classname = self.__obj.__class__.__name__
            except AttributeError: # pragma: no cover
                p.classname = "???"

        def __getattr__(self, key):
            if key.startswith("_VerifyObjectProxy__"): # pragma: no cover
                raise AttributeError
            attr = getattr(self.__obj, key)
            if not callable(attr):
                return VerifyCallProxy(getattr, self.__ns)(self.__obj, key)
            return VerifyCallProxy(attr, self.__ns)

        def __call__(self, *args, **kwargs):
            """Call this to instantiate the type, if a type was passed
            to verify_proxy.  Otherwise, pass the call through."""
            ret = VerifyCallProxy(self.__obj, self.__ns)(*args, **kwargs)
            if type(self.__obj) in (types.TypeType, types.ClassType):
                # Instantiation
                self.__obj = ret
                return self
            return ret

    return VerifyObjectProxy(obj_or_type)
```

Listing B-43: `nilmdb/utils/time.py`: Timestamp parsing and conversion routines. Converts to and from NilmDB representation (integer microseconds since epoch), Unix representation (floating-point seconds since epoch), and free-form human readable representations.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/time.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
from __future__ import absolute_import

from nilmdb.utils import datetime_tz
import re
import time

# Range
min_timestamp = (-2**63)
max_timestamp = (2**63 - 1)

# Smallest representable step
epsilon = 1

def string_to_timestamp(str):
    """Convert a string that represents an integer number of microseconds
    since epoch."""
    try:
        # Parse a string like "1234567890123456" and return an integer
        return int(str)
    except ValueError:
        # Try parsing as a float, in case it's "1234567890123456.0"
        return int(round(float(str)))

def timestamp_to_string(timestamp):
    """Convert a timestamp (integer microseconds since epoch) to a string"""
    if isinstance(timestamp, float):
        return str(int(round(timestamp)))
    else:
        return str(timestamp)

def timestamp_to_human(timestamp):
    """Convert a timestamp (integer microseconds since epoch) to a
    human-readable string, using the local timezone for display
    (e.g. from the TZ env var)."""
    if timestamp == min_timestamp:
        return "(minimum)"
    if timestamp == max_timestamp:
        return "(maximum)"
    dt = datetime_tz.datetime_tz.fromtimestamp(timestamp_to_unix(timestamp))
    return dt.strftime("%a, %d %b %Y %H:%M:%S.%f %z")

def unix_to_timestamp(unix):
    """Convert a Unix timestamp (floating point seconds since epoch)
    into a NILM timestamp (integer microseconds since epoch)"""
    return int(round(unix * 1e6))
seconds_to_timestamp = unix_to_timestamp

def timestamp_to_unix(timestamp):
    """Convert a NILM timestamp (integer microseconds since epoch)
    into a Unix timestamp (floating point seconds since epoch)"""
    return timestamp / 1e6
timestamp_to_seconds = timestamp_to_unix

def rate_to_period(hz, cycles = 1):
```

```
    """Convert a rate (in Hz) to a period (in timestamp units).
    Returns an integer."""
    period = unix_to_timestamp(cycles) / float(hz)
    return int(round(period))

def parse_time(toparse):
    """
    Parse a free-form time string and return a nilmdb timestamp
    (integer microseconds since epoch).  If the string doesn't contain a
    timestamp, the current local timezone is assumed (e.g. from the TZ
    env var).
    """
    if toparse == "min":
        return min_timestamp
    if toparse == "max":
        return max_timestamp

    # If it starts with @, treat it as a NILM timestamp
    # (integer microseconds since epoch)
    try:
        if toparse[0] == '@':
            return int(toparse[1:])
    except (ValueError, KeyError, IndexError):
        pass

    # If string isn't "now" and doesn't contain at least 4 digits,
    # consider it invalid.  smartparse might otherwise accept
    # empty strings and strings with just separators.
    if toparse != "now" and len(re.findall(r"\d", toparse)) < 4:
        raise ValueError("not enough digits for a timestamp")

    # Try to just parse the time as given
    try:
        return unix_to_timestamp(datetime_tz.datetime_tz.
                                 smartparse(toparse).totimestamp())
    except (ValueError, OverflowError):
        pass

    # If it's parseable as a float, treat it as a Unix or NILM
    # timestamp based on its range.
    try:
        val = float(toparse)
        # range is from about year 2001 - 2128
        if val > 1e9 and val < 5e9:
            return unix_to_timestamp(val)
        if val > 1e15 and val < 5e15:
            return val
    except ValueError:
        pass

    # Try to extract a substring in a condensed format that we expect
    # to see in a filename or header comment
    res = re.search(r"(^|[^\d])("            # non-numeric or SOL
                    r"(199\d|2\d\d\d)"       # year
                    r"[-/]?"                 # separator
                    r"(0[1-9]|1[012])"       # month
                    r"[-/]?"                 # separator
                    r"([012]\d|3[01])"       # day
                    r"[-T ]?"                # separator
                    r"([01]\d|2[0-3])"       # hour
                    r"[:]?"                  # separator
                    r"([0-5]\d)"             # minute
                    r"[:]?"                  # separator
                    r"([0-5]\d)?"            # second
                    r"([-+]\d\d\d\d)?"       # timezone
                    r")", toparse)
    if res is not None:
```

```
122         try:
123             return unix_to_timestamp(datetime_tz.datetime_tz.
124                                      smartparse(res.group(2)).totimestamp())
125         except ValueError:
126             pass
127 
128     # Could also try to successively parse substrings, but let's
129     # just give up for now.
130     raise ValueError("unable to parse timestamp")
131 
132 def now():
133     """Return current timestamp"""
134     return unix_to_timestamp(time.time())
```

Listing B-44: `nilmdb/utils/timer.py`: Context manager for block timing. This is used to verify the performance of blocks of code.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/timer.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1  # -*- coding: utf-8 -*-
2  
3  # Simple timer to time a block of code, for optimization debugging
4  # use like:
5  #   with nilmdb.utils.Timer("flush"):
6  #       foo.flush()
7  
8  from __future__ import print_function
9  from __future__ import absolute_import
10 import contextlib
11 import time
12 
13 @contextlib.contextmanager
14 def Timer(name = None, tosyslog = False):
15     start = time.time()
16     yield
17     elapsed = int((time.time() - start) * 1000)
18     msg = (name or 'elapsed') + ": " + str(elapsed) + " ms"
19     if tosyslog: # pragma: no cover
20         import syslog
21         syslog.syslog(msg)
22     else:
23         print(msg)
```

Listing B-45: `nilmdb/utils/timestamper.py`: File object wrapper that adds timestamps to input lines. These classes wrap a file-like object and provide their own file-like interface.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/timestamper.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1  """File-like objects that add timestamps to the input lines"""
2  
3  from nilmdb.utils.printf import *
4  import nilmdb.utils.time
5  
6  class Timestamper(object):
```

```
    """A file-like object that adds timestamps to lines of an input file."""
    def __init__(self, infile, ts_iter):
        """file: filename, or another file-like object
           ts_iter: iterator that returns a timestamp string for
           each line of the file"""
        if isinstance(infile, basestring):
            self.file = open(infile, "r")
        else:
            self.file = infile
        self.ts_iter = ts_iter

    def close(self):
        self.file.close()

    def readline(self, *args):
        while True:
            line = self.file.readline(*args)
            if not line:
                return ""
            if line[0] == '#':
                continue
            break
        try:
            return self.ts_iter.next() + line
        except StopIteration:
            return ""

    def readlines(self, size = None):
        out = ""
        while True:
            line = self.readline()
            out += line
            if not line or (size and len(out) >= size):
                break
        return out

    def __iter__(self):
        return self

    def next(self):
        result = self.readline()
        if not result:
            raise StopIteration
        return result

class TimestamperRate(Timestamper):
    """Timestamper that uses a start time and a fixed rate"""
    def __init__(self, infile, start, rate, end = None):
        """
        file: file name or object

        start: Unix timestamp for the first value

        rate: 1/rate is added to the timestamp for each line

        end: If specified, raise StopIteration before outputting a value
             greater than this."""
        timestamp_to_string = nilmdb.utils.time.timestamp_to_string
        rate_to_period = nilmdb.utils.time.rate_to_period
        def iterator(start, rate, end):
            n = 0
            rate = float(rate)
            while True:
                now = start + rate_to_period(rate, n)
                if end and now >= end:
                    raise StopIteration
                yield timestamp_to_string(now) + " "
```

```
                n += 1
        Timestamper.__init__(self, infile, iterator(start, rate, end))
        self.start = start
        self.rate = rate
    def __str__(self):
        return sprintf("TimestamperRate(..., start=\"%s\", rate=%g)",
                       nilmdb.utils.time.timestamp_to_human(self.start),
                       self.rate)

class TimestamperNow(Timestamper):
    """Timestamper that uses current time"""
    def __init__(self, infile):
        timestamp_to_string = nilmdb.utils.time.timestamp_to_string
        get_now = nilmdb.utils.time.now
        def iterator():
            while True:
                yield timestamp_to_string(get_now()) + " "
        Timestamper.__init__(self, infile, iterator())
    def __str__(self):
        return "TimestamperNow(...)"
```

Listing B-46: `nilmdb/utils/unicode.py`: Unicode string conversion routines. Attempts conversion using the default encoding when necessary, falling back to UTF-8 on failure.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: nilmdb/utils/unicode.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
import sys

if sys.version_info[0] >= 3: # pragma: no cover (future Python3 compat)
    text_type = str
else:
    text_type = unicode

def encode(u):
    """Try to encode something from Unicode to a string using the
    default encoding.  If it fails, try encoding as UTF-8."""
    if not isinstance(u, text_type):
        return u
    try:
        return u.encode()
    except UnicodeEncodeError:
        return u.encode("utf-8")

def decode(s):
    """Try to decode someting from string to Unicode using the
    default encoding.  If it fails, try decoding as UTF-8."""
    if isinstance(s, text_type):
        return s
    try:
        return s.decode()
    except UnicodeDecodeError:
        try:
            return s.decode("utf-8")
        except UnicodeDecodeError:
            return s # best we can do
```

Appendix C

NilmDB Test Suite

This appendix contains the test suite for NilmDB. These tests ensure correctness, validate fixes for previously discovered errors to prevent regressions, and achieve complete coverage of the NilmDB code to ensure against syntax and other runtime-detected errors. Many of the routines here also serve as examples of using NilmDB interfaces from external code.

C.1 Test Utilities

Listing C-1: `tests/runtests.py`: Test fixture plugin to run the tests in a particular order.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/runtests.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  #!/usr/bin/python
2
3  import nose
4  import os
5  import sys
6  import glob
7  from collections import OrderedDict
8
9  # Change into parent dir
10 os.chdir(os.path.dirname(os.path.realpath(__file__)) + "/..")
11
12 class JimOrderPlugin(nose.plugins.Plugin):
13     """When searching for tests and encountering a directory that
14     contains a 'test.order' file, run tests listed in that file, in the
15     order that they're listed.  Globs are OK in that file and duplicates
16     are removed."""
```

```
    name = 'jimorder'
    score = 10000

    def prepareTestLoader(self, loader):
        def wrap(func):
            def wrapper(name, *args, **kwargs):
                addr = nose.selector.TestAddress(
                    name, workingDir=loader.workingDir)
                try:
                    order = os.path.join(addr.filename, "test.order")
                except Exception:
                    order = None
                if order and os.path.exists(order):
                    files = []
                    for line in open(order):
                        line = line.split('#')[0].strip()
                        if not line:
                            continue
                        fn = os.path.join(addr.filename, line.strip())
                        files.extend(sorted(glob.glob(fn)) or [fn])
                    files = list(OrderedDict.fromkeys(files))
                    tests = [ wrapper(fn, *args, **kwargs) for fn in files ]
                    return loader.suiteClass(tests)
                return func(name, *args, **kwargs)
            return wrapper
        loader.loadTestsFromName = wrap(loader.loadTestsFromName)
        return loader

# Use setup.cfg for most of the test configuration.  Adding
# --with-jimorder here means that a normal "nosetests" run will
# still work, it just won't support test.order.
nose.main(addplugins = [ JimOrderPlugin() ],
          argv = sys.argv + ["--with-jimorder"])
```

Listing C-2: `tests/testutil/renderdot.py`: Render graphical visualizations of a binary tree. This assists with debugging interval trees and other aspects of the code.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/testutil/renderdot.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
import sys

class Renderer(object):

    def __init__(self, getleft, getright,
                 getred, getstart, getend, nil):
        self.getleft = getleft
        self.getright = getright
        self.getred = getred
        self.getstart = getstart
        self.getend = getend
        self.nil = nil

    # Rendering
    def __render_dot_node(self, node, max_depth = 20):
        from nilmdb.utils.printf import sprintf
        """Render a single node and its children into a dot graph fragment"""
        if max_depth == 0:
            return ""
        if node is self.nil:
            return ""
```

```
        def c(red):
            if red:
                return 'color="#ff0000", style=filled, fillcolor="#ffc0c0"'
            else:
                return 'color="#000000", style=filled, fillcolor="#c0c0c0"'
        s = sprintf("%d [label=\"%g\\n%g\", %s];\n",
                    id(node),
                    self.getstart(node), self.getend(node),
                    c(self.getred(node)))

        if self.getleft(node) is self.nil:
            s += sprintf("L%d [label=\"-\", %s];\n", id(node), c(False))
            s += sprintf("%d -> L%d [label=L];\n", id(node), id(node))
        else:
            s += sprintf("%d -> %d [label=L];\n",
                         id(node),id(self.getleft(node)))
        if self.getright(node) is self.nil:
            s += sprintf("R%d [label=\"-\", %s];\n", id(node), c(False))
            s += sprintf("%d -> R%d [label=R];\n", id(node), id(node))
        else:
            s += sprintf("%d -> %d [label=R];\n",
                         id(node), id(self.getright(node)))
        s += self.__render_dot_node(self.getleft(node), max_depth-1)
        s += self.__render_dot_node(self.getright(node), max_depth-1)
        return s

    def render_dot(self, rootnode, title = "Tree"):
        """Render the entire tree as a dot graph"""
        return ("digraph rbtree {\n"
                + self.__render_dot_node(rootnode)
                + "}\n");

    def render_dot_live(self, rootnode, title = "Tree"):
        """Render the entiretree as a dot graph, live GTK view"""
        import gtk
        import gtk.gdk
        sys.path.append("/usr/share/xdot")
        import xdot
        xdot.Pen.highlighted = lambda pen: pen
        s = ("digraph rbtree {\n"
             + self.__render_dot_node(rootnode)
             + "}\n");
        window = xdot.DotWindow()
        window.set_dotcode(s)
        window.set_title(title + " - any key to close")
        window.connect('destroy', gtk.main_quit)
        def quit(widget, event):
            if not event.is_modifier:
                window.destroy()
                gtk.main_quit()
        window.widget.connect('key-press-event', quit)
        gtk.main()

class RBTreeRenderer(Renderer):
    def __init__(self, tree):
        Renderer.__init__(self,
                          lambda node: node.left,
                          lambda node: node.right,
                          lambda node: node.red,
                          lambda node: node.start,
                          lambda node: node.end,
                          tree.nil)
        self.tree = tree

    def render(self, title = "RBTree", live = True):
        if live:
            return Renderer.render_dot_live(self, self.tree.getroot(), title)
```

```
89          else:
90              return Renderer.render_dot(self, self.tree.getroot(), title)
```

Listing C-3: `tests/testutil/helpers.py`: Miscellaneous test fixture helpers.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/testutil/helpers.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1   # Just some helpers for test functions
2
3   import shutil, os
4
5   def myrepr(x):
6       if isinstance(x, basestring):
7           return '"' + x + '"'
8       else:
9           return repr(x)
10
11  def eq_(a, b):
12      if not a == b:
13          raise AssertionError("%s != %s" % (myrepr(a), myrepr(b)))
14
15  def lt_(a, b):
16      if not a < b:
17          raise AssertionError("%s is not less than %s" % (myrepr(a), myrepr(b)))
18
19  def in_(a, b):
20      if a not in b:
21          raise AssertionError("%s not in %s" % (myrepr(a), myrepr(b)))
22
23  def in2_(a1, a2, b):
24      if a1 not in b and a2 not in b:
25          raise AssertionError("(%s or %s) not in %s" % (myrepr(a1), myrepr(a2),
26                                                         myrepr(b)))
27
28  def ne_(a, b):
29      if not a != b:
30          raise AssertionError("unexpected %s == %s" % (myrepr(a), myrepr(b)))
31
32  def lines_(a, n):
33      l = a.count('\n')
34      if not l == n:
35          if len(a) > 5000:
36              a = a[0:5000] + " ... truncated"
37          raise AssertionError("wanted %d lines, got %d in output: '%s'"
38                               % (n, l, a))
39
40  def recursive_unlink(path):
41      try:
42          shutil.rmtree(path)
43      except OSError:
44          pass
45      try:
46          os.unlink(path)
47      except OSError:
48          pass
```

C.2 Tests

Listing C-4: `tests/test_printf.py`: Test the `*printf` series of functions.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_printf.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  import nilmdb
2  from nilmdb.utils.printf import *
3
4  from nose.tools import *
5  from nose.tools import assert_raises
6  from cStringIO import StringIO
7  import sys
8
9  from testutil.helpers import *
10
11 class TestPrintf(object):
12     def test_printf(self):
13         old_stdout = sys.stdout
14         sys.stdout = test1 = StringIO()
15         test2 = StringIO()
16         test3 = ""
17         try:
18             printf("hello, world: %d", 123)
19             fprintf(test2, "hello too: %d", 123)
20             test3 = sprintf("hello three: %d", 123)
21         except Exception:
22             sys.stdout = old_stdout
23             raise
24         sys.stdout = old_stdout
25         eq_(test1.getvalue(), "hello, world: 123")
26         eq_(test2.getvalue(), "hello too: 123")
27         eq_(test3, "hello three: 123")
```

Listing C-5: `tests/test_threadsafety.py`: Test the thread safety verification proxy.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_threadsafety.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  import nilmdb
2  from nilmdb.utils.printf import *
3
4  import nose
5  from nose.tools import *
6  from nose.tools import assert_raises
7
8  from testutil.helpers import *
9  import threading
10
11 class Thread(threading.Thread):
12     def __init__(self, target):
13         self.target = target
14         threading.Thread.__init__(self)
15
16     def run(self):
```

```
        try:
            self.target()
        except AssertionError as e:
            self.error = e
        else:
            self.error = None

class Test():
    def __init__(self):
        self.test = 1234

    @classmethod
    def asdf(cls):
        pass

    def foo(self, exception = False, reenter = False):
        if exception:
            raise Exception()
        self.bar(reenter)

    def bar(self, reenter):
        if reenter:
            self.foo()
        return 123

    def baz_threaded(self, target):
        t = Thread(target)
        t.start()
        t.join()
        return t

    def baz(self, target):
        target()

class TestThreadSafety(object):
    def tryit(self, c, threading_ok, concurrent_ok):
        eq_(c.test, 1234)
        c.foo()
        t = Thread(c.foo)
        t.start()
        t.join()
        if threading_ok and t.error:
            raise Exception("got unexpected error: " + str(t.error))
        if not threading_ok and not t.error:
            raise Exception("failed to get expected error")
        try:
            c.baz(c.foo)
        except AssertionError as e:
            if concurrent_ok:
                raise Exception("got unexpected error: " + str(e))
        else:
            if not concurrent_ok:
                raise Exception("failed to get expected error")
        t = c.baz_threaded(c.foo)
        if (concurrent_ok and threading_ok) and t.error:
            raise Exception("got unexpected error: " + str(t.error))
        if not (concurrent_ok and threading_ok) and not t.error:
            raise Exception("failed to get expected error")

    def test(self):
        proxy = nilmdb.utils.threadsafety.verify_proxy
        self.tryit(Test(), True, True)
        self.tryit(proxy(Test(), True, True, True), False, False)
        self.tryit(proxy(Test(), True, True, False), False, True)
        self.tryit(proxy(Test(), True, False, True), True, False)
        self.tryit(proxy(Test(), True, False, False), True, True)
        self.tryit(proxy(Test, True, True, True)(), False, False)
```

```
84          self.tryit(proxy(Test, True, True, False)(), False, True)
85          self.tryit(proxy(Test, True, False, True)(), True, False)
86          self.tryit(proxy(Test, True, False, False)(), True, True)
87
88          proxy(proxy(proxy(Test))()).foo()
89
90          c = proxy(Test())
91          c.foo()
92          try:
93              c.foo(exception = True)
94          except Exception:
95              pass
96          c.foo()
```

Listing C-6: `tests/test_lrucache.py`: Test fill, cache, and evict behavior of LRU caches.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_lrucache.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1   import nilmdb
2   from nilmdb.utils.printf import *
3
4   import nose
5   from nose.tools import *
6   from nose.tools import assert_raises
7   import threading
8   import time
9   import inspect
10
11  from testutil.helpers import *
12
13  @nilmdb.utils.lru_cache(size = 3)
14  def foo1(n):
15      return n
16
17  @nilmdb.utils.lru_cache(size = 5)
18  def foo2(n):
19      return n
20
21  def foo3d(n):
22      foo3d.destructed.append(n)
23  foo3d.destructed = []
24  @nilmdb.utils.lru_cache(size = 3, onremove = foo3d)
25  def foo3(n):
26      return n
27
28  class Foo:
29      def __init__(self):
30          self.calls = 0
31      @nilmdb.utils.lru_cache(size = 3, keys = slice(1, 2))
32      def foo(self, n, **kwargs):
33          self.calls += 1
34
35  class TestLRUCache(object):
36      def test(self):
37
38          [ foo1(n) for n in [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ] ]
39          eq_(foo1.cache_info(), (6, 3))
40          [ foo1(n) for n in [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ] ]
41          eq_(foo1.cache_info(), (15, 3))
42          [ foo1(n) for n in [ 4, 2, 1, 1, 4 ] ]
```

```
43          eq_(foo1.cache_info(), (18, 5))
44
45          [ foo2(n) for n in [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ] ]
46          eq_(foo2.cache_info(), (6, 3))
47          [ foo2(n) for n in [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ] ]
48          eq_(foo2.cache_info(), (15, 3))
49          [ foo2(n) for n in [ 4, 2, 1, 1, 4 ] ]
50          eq_(foo2.cache_info(), (19, 4))
51
52          [ foo3(n) for n in [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ] ]
53          eq_(foo3.cache_info(), (6, 3))
54          [ foo3(n) for n in [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ] ]
55          eq_(foo3.cache_info(), (15, 3))
56          [ foo3(n) for n in [ 4, 2, 1, 1, 4 ] ]
57          eq_(foo3.cache_info(), (18, 5))
58          eq_(foo3d.destructed, [1, 3])
59          with assert_raises(KeyError):
60              foo3.cache_remove(1,2,3)
61          foo3.cache_remove(1)
62          eq_(foo3d.destructed, [1, 3, 1])
63          foo3.cache_remove_all()
64          eq_(foo3d.destructed, [1, 3, 1, 2, 4 ])
65
66          foo = Foo()
67          foo.foo(5)
68          foo.foo(6)
69          foo.foo(7)
70          foo.foo(5)
71          eq_(foo.calls, 3)
72
73          # Can't handle keyword arguments right now
74          with assert_raises(NotImplementedError):
75              foo.foo(3, asdf = 7)
76
77          # Verify that argspecs were maintained
78          eq_(inspect.getargspec(foo1),
79              inspect.ArgSpec(args=['n'],
80                              varargs=None, keywords=None, defaults=None))
81          eq_(inspect.getargspec(foo.foo),
82              inspect.ArgSpec(args=['self', 'n'],
83                              varargs=None, keywords="kwargs", defaults=None))
```

Listing C-7: `tests/test_mustclose.py`: Test the `must_close` decorator.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_mustclose.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1   import nilmdb
2   from nilmdb.utils.printf import *
3
4   import nose
5   from nose.tools import *
6   from nose.tools import assert_raises
7
8   from testutil.helpers import *
9
10  import sys
11  import cStringIO
12  import gc
13
14  import inspect
15
16  err = cStringIO.StringIO()
```

```
17
18  @nilmdb.utils.must_close(errorfile = err)
19  class Foo:
20      def __init__(self, arg):
21          fprintf(err, "Init %s\n", arg)
22
23      def __del__(self):
24          fprintf(err, "Deleting\n")
25
26      def close(self):
27          fprintf(err, "Closing\n")
28
29  @nilmdb.utils.must_close(errorfile = err, wrap_verify = True)
30  class Bar:
31      def __init__(self):
32          fprintf(err, "Init\n")
33
34      def __del__(self):
35          fprintf(err, "Deleting\n")
36
37      @classmethod
38      def baz(self):
39          fprintf(err, "Baz\n")
40
41      def close(self):
42          fprintf(err, "Closing\n")
43
44      def blah(self, arg):
45          fprintf(err, "Blah %s\n", arg)
46
47  @nilmdb.utils.must_close(errorfile = err)
48  class Baz:
49      pass
50
51  class TestMustClose(object):
52      def test(self):
53
54          # Note: this test might fail if the Python interpreter doesn't
55          # garbage collect the object (and call its __del__ function)
56          # right after a "del x".
57
58          # Trigger error
59          err.truncate()
60          x = Foo("hi")
61          # Verify that the arg spec was maintained
62          eq_(inspect.getargspec(x.__init__),
63              inspect.ArgSpec(args = ['self', 'arg'],
64                              varargs = None, keywords = None, defaults = None))
65          del x
66          gc.collect()
67          eq_(err.getvalue(),
68              "Init hi\n"
69              "error: Foo.close() wasn't called!\n"
70              "Deleting\n")
71
72          # No error
73          err.truncate(0)
74          y = Foo("bye")
75          y.close()
76          del y
77          gc.collect()
78          eq_(err.getvalue(),
79              "Init bye\n"
80              "Closing\n"
81              "Deleting\n")
82
83          # Verify function calls when wrap_verify is True
```

```
84          err.truncate(0)
85          z = Bar()
86          eq_(inspect.getargspec(z.blah),
87              inspect.ArgSpec(args = ['self', 'arg'],
88                              varargs = None, keywords = None, defaults = None))
89          z.blah("boo")
90          z.close()
91          with assert_raises(AssertionError) as e:
92              z.blah("hello")
93          in_("called <function blah at 0x", str(e.exception))
94          in_("> after close", str(e.exception))
95          # Since the most recent assertion references 'z',
96          # we need to raise another assertion here so that
97          # 'z' will get properly deleted.
98          with assert_raises(AssertionError):
99              raise AssertionError()
100         del z
101         gc.collect()
102         eq_(err.getvalue(),
103             "Init\n"
104             "Blah boo\n"
105             "Closing\n"
106             "Deleting\n")
107
108         # Class with missing methods
109         err.truncate(0)
110         w = Baz()
111         w.close()
112         del w
113         eq_(err.getvalue(), "")
```

Listing C-8: `tests/test_serializer.py`: Test the multithreaded method call serializer.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/test_serializer.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   import nilmdb
2   from nilmdb.utils.printf import *
3
4   import nose
5   from nose.tools import *
6   from nose.tools import assert_raises
7   import threading
8   import time
9
10  from testutil.helpers import *
11
12  class Foo(object):
13      val = 0
14
15      def __init__(self, asdf = "asdf"):
16          self.init_thread = threading.current_thread().name
17
18      @classmethod
19      def foo(self):
20          pass
21
22      def fail(self):
23          raise Exception("you asked me to do this")
24
25      def test(self, debug = False):
```

```
        self.tester(debug)

    def t(self):
        pass

    def tester(self, debug = False):
        # purposely not thread-safe
        self.test_thread = threading.current_thread().name
        oldval = self.val
        newval = oldval + 1
        time.sleep(0.05)
        self.val = newval
        if debug:
            printf("[%s] value changed: %d -> %d\n",
                   threading.current_thread().name, oldval, newval)

class Base(object):

    def test_wrapping(self):
        self.foo.test()
        with assert_raises(Exception):
            self.foo.fail()

    def test_threaded(self):
        def func(foo):
            foo.test()
        threads = []
        for i in xrange(20):
            threads.append(threading.Thread(target = func, args = (self.foo,)))
        for t in threads:
            t.start()
        for t in threads:
            t.join()
        self.verify_result()

    def verify_result(self):
        eq_(self.foo.val, 20)
        eq_(self.foo.init_thread, self.foo.test_thread)

class ListLike(object):
    def __init__(self):
        self.thread = threading.current_thread().name
        self.foo = 0

    def __iter__(self):
        eq_(threading.current_thread().name, self.thread)
        self.foo = 0
        return self

    def __getitem__(self, key):
        eq_(threading.current_thread().name, self.thread)
        return key

    def next(self):
        eq_(threading.current_thread().name, self.thread)
        if self.foo < 5:
            self.foo += 1
            return self.foo
        else:
            raise StopIteration

class TestUnserialized(Base):
    def setUp(self):
        self.foo = Foo()

    def verify_result(self):
        # This should have failed to increment properly
```

```
93          ne_(self.foo.val, 20)
94          # Init and tests ran in different threads
95          ne_(self.foo.init_thread, self.foo.test_thread)
96
97  class TestSerializer(Base):
98      def setUp(self):
99          self.foo = nilmdb.utils.serializer_proxy(Foo)("qwer")
100
101     def test_multi(self):
102         sp = nilmdb.utils.serializer_proxy
103         sp(Foo("x")).t()
104         sp(sp(Foo)("x")).t()
105         sp(sp(Foo))("x").t()
106         sp(sp(Foo("x"))).t()
107         sp(sp(Foo)("x")).t()
108         sp(sp(Foo))("x").t()
109
110     def test_iter(self):
111         sp = nilmdb.utils.serializer_proxy
112         i = sp(ListLike)()
113         eq_(list(i), [1,2,3,4,5])
114         eq_(i[3], 3)
```

Listing C-9: `tests/test_timestamper.py`: Test ASCII text timestamper.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/test_timestamper.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1   import nilmdb
2   from nilmdb.utils.printf import *
3
4   from nose.tools import *
5   from nose.tools import assert_raises
6   import os
7   import sys
8   import cStringIO
9
10  from testutil.helpers import *
11
12  from nilmdb.utils import timestamper
13
14  class TestTimestamper(object):
15
16      # Not a very comprehensive test, but it's good enough.
17
18      def test_timestamper(self):
19          def join(list):
20              return "\n".join(list) + "\n"
21
22          start = nilmdb.utils.time.parse_time("03/24/2012")
23          lines_in  = [ "hello", "world", "hello world", "# commented out" ]
24          lines_out = [ "1332561600000000 hello",
25                        "1332561600000125 world",
26                        "1332561600000250 hello world" ]
27
28          # full
29          input = cStringIO.StringIO(join(lines_in))
30          ts = timestamper.TimestamperRate(input, start, 8000)
31          foo = ts.readlines()
32          eq_(foo, join(lines_out))
33          in_("TimestamperRate(..., start=", str(ts))
34
35          # first 30 or so bytes means the first 2 lines
```

```
36          input = cStringIO.StringIO(join(lines_in))
37          ts = timestamper.TimestamperRate(input, start, 8000)
38          foo = ts.readlines(30)
39          eq_(foo, join(lines_out[0:2]))
40
41          # stop iteration early
42          input = cStringIO.StringIO(join(lines_in))
43          ts = timestamper.TimestamperRate(input, start, 8000,
44                                           1332561600000200)
45          foo = ""
46          for line in ts:
47              foo += line
48          eq_(foo, join(lines_out[0:2]))
49
50          # stop iteration early (readlines)
51          input = cStringIO.StringIO(join(lines_in))
52          ts = timestamper.TimestamperRate(input, start, 8000,
53                                           1332561600000200)
54          foo = ts.readlines()
55          eq_(foo, join(lines_out[0:2]))
56
57          # stop iteration really early
58          input = cStringIO.StringIO(join(lines_in))
59          ts = timestamper.TimestamperRate(input, start, 8000,
60                                           1332561600000000)
61          foo = ts.readlines()
62          eq_(foo, "")
63
64          # use iterator
65          input = cStringIO.StringIO(join(lines_in))
66          ts = timestamper.TimestamperRate(input, start, 8000)
67          foo = ""
68          for line in ts:
69              foo += line
70          eq_(foo, join(lines_out))
71
72          # check that TimestamperNow gives similar result
73          input = cStringIO.StringIO(join(lines_in))
74          ts = timestamper.TimestamperNow(input)
75          foo = ts.readlines()
76          ne_(foo, join(lines_out))
77          eq_(len(foo), len(join(lines_out)))
78          eq_(str(ts), "TimestamperNow(...)")
79
80          # Test passing a file (should be empty)
81          ts = timestamper.TimestamperNow("/dev/null")
82          for line in ts:
83              raise AssertionError
84          ts.close()
```

Listing C-10: `tests/test_rbtree.py`: Test red-black tree implementation.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_rbtree.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
1  # -*- coding: utf-8 -*-
2
3  import nilmdb
4  from nilmdb.utils.printf import *
5
6  from nose.tools import *
7  from nose.tools import assert_raises
8
```

```
from nilmdb.server.rbtree import RBTree, RBNode

from testutil.helpers import *
import unittest

# set to False to skip live renders
do_live_renders = False
def render(tree, description = "", live = True):
    import testutil.renderdot as renderdot
    r = renderdot.RBTreeRenderer(tree)
    return r.render(description, live and do_live_renders)

class TestRBTree:
    def test_rbtree(self):
        rb = RBTree()
        rb.insert(RBNode(10000, 10001))
        rb.insert(RBNode(10004, 10007))
        rb.insert(RBNode(10001, 10002))
        # There was a typo that gave the RBTree a loop in this case.
        # Verify that the dot isn't too big.
        s = render(rb, live = False)
        assert(len(s.splitlines()) < 30)

    def test_rbtree_big(self):
        import random
        random.seed(1234)

        # make a set of 100 intervals, inserted in order
        rb = RBTree()
        j = 100
        for i in xrange(j):
            rb.insert(RBNode(i, i+1))
        render(rb, "in-order insert")

        # remove about half of them
        for i in random.sample(xrange(j),j):
            if random.randint(0,1):
                rb.delete(rb.find(i, i+1))
        render(rb, "in-order insert, random delete")

        # make a set of 100 intervals, inserted at random
        rb = RBTree()
        j = 100
        for i in random.sample(xrange(j),j):
            rb.insert(RBNode(i, i+1))
        render(rb, "random insert")

        # remove about half of them
        for i in random.sample(xrange(j),j):
            if random.randint(0,1):
                rb.delete(rb.find(i, i+1))
        render(rb, "random insert, random delete")

        # in-order insert of 50 more
        for i in xrange(50):
            rb.insert(RBNode(i+500, i+501))
        render(rb, "random insert, random delete, in-order insert")

    def test_rbtree_basics(self):
        rb = RBTree()
        vals = [ 7, 14, 1, 2, 8, 11, 5, 15, 4]
        for n in vals:
            rb.insert(RBNode(n, n))

        # stringify
        s = ""
        for node in rb:
```

```
            s += str(node)
        in_("[node (None) 1", s)
        eq_(str(rb.nil), "[node nil]")

        # inorder traversal, successor and predecessor
        last = 0
        for node in rb:
            assert(node.start > last)
            last = node.start
            successor = rb.successor(node)
            if successor:
                assert(rb.predecessor(successor) is node)
            predecessor = rb.predecessor(node)
            if predecessor:
                assert(rb.successor(predecessor) is node)

        # Delete node not in the tree
        with assert_raises(AttributeError):
            rb.delete(RBNode(1,2))

        # Delete all nodes!
        for node in rb:
            rb.delete(node)

        # Build it up again, make sure it matches
        for n in vals:
            rb.insert(RBNode(n, n))
        s2 = ""
        for node in rb:
            s2 += str(node)
        assert(s == s2)

    def test_rbtree_find(self):
        # Get a little bit of coverage for some overlapping cases,
        # even though the class doesn't fully support it.
        rb = RBTree()
        nodes = [ RBNode(1, 5), RBNode(1, 10), RBNode(1, 15) ]
        for n in nodes:
            rb.insert(n)
        assert(rb.find(1, 5) is nodes[0])
        assert(rb.find(1, 10) is nodes[1])
        assert(rb.find(1, 15) is nodes[2])

    def test_rbtree_find_leftright(self):
        # Now let's get some ranges in there
        rb = RBTree()
        vals = [ 7, 14, 1, 2, 8, 11, 5, 15, 4]
        for n in vals:
            rb.insert(RBNode(n*10, n*10+5))

        # Check find_end_left, find_right_start
        for i in range(160):
            left = rb.find_left_end(i)
            right = rb.find_right_start(i)
            if left:
                # endpoint should be more than i
                assert(left.end >= i)
                # all earlier nodes should have a lower endpoint
                for node in rb:
                    if node is left:
                        break
                    assert(node.end < i)
            if right:
                # startpoint should be less than i
                assert(right.start <= i)
                # all later nodes should have a higher startpoint
                for node in reversed(list(rb)):
```

```
                    if node is right:
                        break
                    assert(node.start > i)

    def test_rbtree_intersect(self):
        # Fill with some ranges
        rb = RBTree()
        rb.insert(RBNode(10,20))
        rb.insert(RBNode(20,25))
        rb.insert(RBNode(30,40))
        # Just a quick test; test_interval will do better.
        eq_(len(list(rb.intersect(1,100))), 3)
        eq_(len(list(rb.intersect(10,20))), 1)
        eq_(len(list(rb.intersect(5,15))), 1)
        eq_(len(list(rb.intersect(15,15))), 1)
        eq_(len(list(rb.intersect(20,21))), 1)
        eq_(len(list(rb.intersect(19,21))), 2)
```

Listing C-11: `tests/test_interval.py`: Test interval management, operations, and performance.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/test_interval.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
# -*- coding: utf-8 -*-

import nilmdb
from nilmdb.utils.printf import *
from nilmdb.utils import datetime_tz

from nose.tools import *
from nose.tools import assert_raises
import itertools

from nilmdb.utils.interval import IntervalError
from nilmdb.server.interval import Interval, DBInterval, IntervalSet

# so we can test them separately
from nilmdb.utils.interval import Interval as UtilsInterval

from testutil.helpers import *
import unittest

# set to False to skip live renders
do_live_renders = False
def render(iset, description = "", live = True):
    import testutil.renderdot as renderdot
    r = renderdot.RBTreeRenderer(iset.tree)
    return r.render(description, live and do_live_renders)

def makeset(string):
    """Build an IntervalSet from a string, for testing purposes

    Each character is 1 second
    [ = interval start
    | = interval end + next start
    ] = interval end
    . = zero-width interval (identical start and end)
    anything else is ignored
    """
    iset = IntervalSet()
    for i, c in enumerate(string):
```

```
        day = i + 10000
        if (c == "["):
            start = day
        elif (c == "|"):
            iset += Interval(start, day)
            start = day
        elif (c == ")"):
            iset += Interval(start, day)
            del start
        elif (c == "."):
            iset += Interval(day, day)
    return iset

class TestInterval:
    def test_client_interval(self):
        # Run interval tests against the Python version of Interval.
        global Interval
        NilmdbInterval = Interval
        Interval = UtilsInterval
        self.test_interval()
        self.test_interval_intersect()
        Interval = NilmdbInterval

        # Other helpers in nilmdb.utils.interval
        i = [ UtilsInterval(1,2), UtilsInterval(2,3), UtilsInterval(4,5) ]
        eq_(list(nilmdb.utils.interval.optimize(i)),
            [ UtilsInterval(1,3), UtilsInterval(4,5) ])
        eq_(UtilsInterval(1234567890123456, 1234567890654321).human_string(),
            "[ Fri, 13 Feb 2009 18:31:30.123456 -0500 -> " +
            "Fri, 13 Feb 2009 18:31:30.654321 -0500 ]")

    def test_interval(self):
        # Test Interval class
        os.environ['TZ'] = "America/New_York"
        datetime_tz._localtz = None
        (d1, d2, d3) = [ nilmdb.utils.time.parse_time(x)
                         for x in [ "03/24/2012", "03/25/2012", "03/26/2012" ] ]

        # basic construction
        i = Interval(d1, d2)
        i = Interval(d1, d3)
        eq_(i.start, d1)
        eq_(i.end, d3)

        # assignment is allowed, but not verified
        i.start = d2
        #with assert_raises(IntervalError):
        #    i.end = d1
        i.start = d1
        i.end = d2

        # end before start
        with assert_raises(IntervalError):
            i = Interval(d3, d1)

        # compare
        assert(Interval(d1, d2) == Interval(d1, d2))
        assert(Interval(d1, d2) < Interval(d1, d3))
        assert(Interval(d1, d3) > Interval(d1, d2))
        assert(Interval(d1, d2) < Interval(d2, d3))
        assert(Interval(d1, d3) < Interval(d2, d3))
        assert(Interval(d2, d2+1) > Interval(d1, d3))
        assert(Interval(d3, d3+1) == Interval(d3, d3+1))
        #with assert_raises(TypeError): # was AttributeError, that's wrong
        #    x = (i == 123)

        # subset
```

```
        eq_(Interval(d1, d3).subset(d1, d2), Interval(d1, d2))
        with assert_raises(IntervalError):
            x = Interval(d2, d3).subset(d1, d2)

        # big integers, negative integers
        x = Interval(5000111222000000, 6000111222000000)
        eq_(str(x), "[5000111222000000 -> 6000111222000000)")
        x = Interval(-5000111222000000, -4000111222000000)
        eq_(str(x), "[-5000111222000000 -> -4000111222000000)")

        # misc
        i = Interval(d1, d2)
        eq_(repr(i), repr(eval(repr(i))))
        eq_(str(i), "[1332561600000000 -> 1332648000000000)")

    def test_interval_intersect(self):
        # Test Interval intersections
        dates = [ 100, 200, 300, 400 ]
        perm = list(itertools.permutations(dates, 2))
        prod = list(itertools.product(perm, perm))
        should_intersect = {
            False: [4, 5, 8, 20, 48, 56, 60, 96, 97, 100],
            True: [0, 1, 2, 12, 13, 14, 16, 17, 24, 25, 26, 28, 29,
                   32, 49, 50, 52, 53, 61, 62, 64, 65, 68, 98, 101, 104]
            }
        for i,((a,b),(c,d)) in enumerate(prod):
            try:
                i1 = Interval(a, b)
                i2 = Interval(c, d)
                eq_(i1.intersects(i2), i2.intersects(i1))
                in_(i, should_intersect[i1.intersects(i2)])
            except IntervalError:
                assert(i not in should_intersect[True] and
                       i not in should_intersect[False])
        with assert_raises(TypeError):
            x = i1.intersects(1234)

    def test_intervalset_construct(self):
        # Test IntervalSet construction
        dates = [ 100, 200, 300, 400 ]

        a = Interval(dates[0], dates[1])
        b = Interval(dates[1], dates[2])
        c = Interval(dates[0], dates[2])
        d = Interval(dates[2], dates[3])

        iseta = IntervalSet(a)
        isetb = IntervalSet([a, b])
        isetc = IntervalSet([a])
        ne_(iseta, isetb)
        eq_(iseta, isetc)
        with assert_raises(TypeError):
            x = iseta != 3
        ne_(IntervalSet(a), IntervalSet(b))

        # Note that assignment makes a new reference (not a copy)
        isetd = IntervalSet(isetb)
        isete = isetd
        eq_(isetd, isetb)
        eq_(isetd, isete)
        isetd -= a
        ne_(isetd, isetb)
        eq_(isetd, isete)

        # test iterator
        for interval in iseta:
            pass
```

```
173
174         # overlap
175         with assert_raises(IntervalError):
176             x = IntervalSet([a, b, c])
177
178         # bad types
179         with assert_raises(Exception):
180             x = IntervalSet([1, 2])
181
182         iset = IntervalSet(isetb)   # test iterator
183         eq_(iset, isetb)
184         eq_(len(iset), 2)
185         eq_(len(IntervalSet()), 0)
186
187         # Test adding
188         iset = IntervalSet(a)
189         iset += IntervalSet(b)
190         eq_(iset, IntervalSet([a, b]))
191
192         iset = IntervalSet(a)
193         iset += b
194         eq_(iset, IntervalSet([a, b]))
195
196         iset = IntervalSet(a)
197         iset.iadd_nocheck(b)
198         eq_(iset, IntervalSet([a, b]))
199
200         iset = IntervalSet(a) + IntervalSet(b)
201         eq_(iset, IntervalSet([a, b]))
202
203         iset = IntervalSet(b) + a
204         eq_(iset, IntervalSet([a, b]))
205
206         # A set consisting of [0-1],[1-2] should match a set consisting of [0-2]
207         eq_(IntervalSet([a,b]), IntervalSet([c]))
208         # Etc
209         ne_(IntervalSet([a,d]), IntervalSet([c]))
210         ne_(IntervalSet([c]), IntervalSet([a,d]))
211         ne_(IntervalSet([c,d]), IntervalSet([b,d]))
212
213         # misc
214         eq_(repr(iset), repr(eval(repr(iset))))
215         eq_(str(iset),
216             "[[100 -> 200), [200 -> 300)]")
217
218     def test_intervalset_geniset(self):
219         # Test basic iset construction
220         eq_(makeset("  [----)   "),
221             makeset("  [-|--)   "))
222
223         eq_(makeset("[)  [--)   ") +
224             makeset(" [)    [--)"),
225             makeset("[|) [-----)"))
226
227         eq_(makeset("  [-------)"),
228             makeset("  [-|-----|"))
229
230
231     def test_intervalset_intersect_difference(self):
232         # Test intersection (&)
233         with assert_raises(TypeError): # was AttributeError
234             x = makeset("[--)") & 1234
235
236         def do_test(a, b, c, d):
237             # a & b == c (using nilmdb.server.interval)
238             ab = IntervalSet()
239             for x in b:
```

```
                for i in (a & x):
                    ab += i
            eq_(ab,c)

            # a & b == c (using nilmdb.utils.interval)
            eq_(IntervalSet(nilmdb.utils.interval.intersection(a,b)), c)

            # a \ b == d
            eq_(IntervalSet(nilmdb.utils.interval.set_difference(a,b)), d)

        # Intersection with intervals
        do_test(makeset("[---|---)[)"),
                makeset("  [------) "),
                makeset("  [-----)  "), # intersection
                makeset("[-)      [)")) # difference

        do_test(makeset("[---------)"),
                makeset(" [---)     "),
                makeset(" [---)     "), # intersection
                makeset("[)   [----)")) # difference

        do_test(makeset(" [---)     "),
                makeset("[---------)"),
                makeset(" [---)     "), # intersection
                makeset("           ")) # difference

        do_test(makeset("    [-----)"),
                makeset(" [-----)   "),
                makeset("    [--)   "), # intersection
                makeset("       [--)")) # difference

        do_test(makeset(" [--)  [--)"),
                makeset("  [------) "),
                makeset("  [-)  [-) "), # intersection
                makeset(" [)      [)")) # difference

        do_test(makeset("      [---)"),
                makeset(" [--)      "),
                makeset("           "), # intersection
                makeset("      [---)")) # difference

        do_test(makeset("    [-|---)"),
                makeset(" [-----|-) "),
                makeset("    [----) "), # intersection
                makeset("         [)")) # difference

        do_test(makeset("    [-|-)  "),
                makeset(" [-|--|--) "),
                makeset("    [---)  "), # intersection
                makeset("           ")) # difference

        do_test(makeset("[-)[-)[-)[)"),
                makeset(" [)  [|)[) "),
                makeset(" [)   [)   "), # intersection
                makeset("[) [-) [)[)")) # difference

        # Border cases -- will give different results if intervals are
        # half open or fully closed.  In nilmdb, they are half open.
        do_test(makeset("      [---)"),
                makeset(" [----)    "),
                makeset("           "), # intersection
                makeset("      [---)")) # difference

        do_test(makeset(" [----)[--)"),
                makeset("[-) [--) [)"),
                makeset(" [) [-)  [)"), # intersection
                makeset("  [-)  [-) ")) # difference
```

```
307
308         # Set difference with bounds
309         a = makeset(" [----)[--)")
310         b = makeset("[-) [--) [)")
311         c = makeset("[----)     ")
312         d = makeset("  [-)      ")
313         eq_(nilmdb.utils.interval.set_difference(
314             a.intersection(list(c)[0]), b.intersection(list(c)[0])), d)
315
316         # Fill out test coverage for non-subsets
317         def diff2(a,b, subset):
318             return nilmdb.utils.interval._interval_math_helper(
319                 a, b, (lambda a, b: b and not a), subset=subset)
320         with assert_raises(nilmdb.utils.interval.IntervalError):
321             list(diff2(a,b,True))
322         list(diff2(a,b,False))
323
324         # Empty second set
325         eq_(nilmdb.utils.interval.set_difference(a, IntervalSet()), a)
326
327         # Empty second set
328         eq_(nilmdb.utils.interval.set_difference(a, IntervalSet()), a)
329
330 class TestIntervalDB:
331     def test_dbinterval(self):
332         # Test DBInterval class
333         i = DBInterval(100, 200, 100, 200, 10000, 20000)
334         eq_(i.start, 100)
335         eq_(i.end, 200)
336         eq_(i.db_start, 100)
337         eq_(i.db_end, 200)
338         eq_(i.db_startpos, 10000)
339         eq_(i.db_endpos, 20000)
340         eq_(repr(i), repr(eval(repr(i))))
341
342         # end before start
343         with assert_raises(IntervalError):
344             i = DBInterval(200, 100, 100, 200, 10000, 20000)
345
346         # db_start too late
347         with assert_raises(IntervalError):
348             i = DBInterval(100, 200, 150, 200, 10000, 20000)
349
350         # db_end too soon
351         with assert_raises(IntervalError):
352             i = DBInterval(100, 200, 100, 150, 10000, 20000)
353
354         # actual start, end can be a subset
355         a = DBInterval(150, 200, 100, 200, 10000, 20000)
356         b = DBInterval(100, 150, 100, 200, 10000, 20000)
357         c = DBInterval(150, 160, 100, 200, 10000, 20000)
358
359         # Make a set of DBIntervals
360         iseta = IntervalSet([a, b])
361         isetc = IntervalSet(c)
362         assert(iseta.intersects(a))
363         assert(iseta.intersects(b))
364
365         # Test subset
366         with assert_raises(IntervalError):
367             x = a.subset(150, 250)
368
369         # Subset of those IntervalSets should still contain DBIntervals
370         for i in IntervalSet(iseta.intersection(Interval(125,250))):
371             assert(isinstance(i, DBInterval))
372
373 class TestIntervalTree:
```

```
374
375     def test_interval_tree(self):
376         import random
377         random.seed(1234)
378
379         # make a set of 100 intervals
380         iset = IntervalSet()
381         j = 100
382         for i in random.sample(xrange(j),j):
383             interval = Interval(i, i+1)
384             iset += interval
385         render(iset, "Random Insertion")
386
387         # remove about half of them
388         for i in random.sample(xrange(j),j):
389             if random.randint(0,1):
390                 iset -= Interval(i, i+1)
391
392         # try removing an interval that doesn't exist
393         with assert_raises(IntervalError):
394             iset -= Interval(1234,5678)
395         render(iset, "Random Insertion, deletion")
396
397         # make a set of 100 intervals, inserted in order
398         iset = IntervalSet()
399         j = 100
400         for i in xrange(j):
401             interval = Interval(i, i+1)
402             iset += interval
403         render(iset, "In-order insertion")
404
405 class TestIntervalSpeed:
406     @unittest.skip("this is slow")
407     def test_interval_speed(self):
408         import yappi
409         import time
410         import random
411         import math
412
413         print
414         yappi.start()
415         speeds = {}
416         limit = 22 # was 20
417         for j in [ 2**x for x in range(5,limit) ]:
418             start = time.time()
419             iset = IntervalSet()
420             for i in random.sample(xrange(j),j):
421                 interval = Interval(i, i+1)
422                 iset += interval
423             speed = (time.time() - start) * 1000000.0
424             printf("%d: %g µs (%g µs each, O(n log n) ratio %g)\n",
425                    j,
426                    speed,
427                    speed/j,
428                    speed / (j*math.log(j))) # should be constant
429             speeds[j] = speed
430         yappi.stop()
431         yappi.print_stats(sort_type=yappi.SORTTYPE_TTOT, limit=10)
```

Listing C-12: `tests/test_bulkdata.py`: Test low-level bulkdata storage interface.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/test_bulkdata.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
# -*- coding: utf-8 -*-

import nilmdb
from nilmdb.utils.printf import *
from nose.tools import *
from nose.tools import assert_raises
import itertools

from testutil.helpers import *

testdb = "tests/bulkdata-testdb"

import nilmdb.server.bulkdata
from nilmdb.server.bulkdata import BulkData

class TestBulkData(object):

    def test_bulkdata(self):
        for (size, files, db) in [ ( 0, 0, testdb ),
                                   ( 25, 1000, testdb ),
                                   ( 1000, 3, testdb.decode("utf-8") ) ]:
            recursive_unlink(db)
            os.mkdir(db)
            self.do_basic(db, size, files)

    def do_basic(self, db, size, files):
        """Do the basic test with variable file_size and files_per_dir"""
        if not size or not files:
            data = BulkData(db)
        else:
            data = BulkData(db, file_size = size, files_per_dir = files)

        # Try opening it again (should result in locking error)
        with assert_raises(IOError) as e:
            data2 = BulkData(db)
        in_("already locked by another process", str(e.exception))

        # create empty
        with assert_raises(ValueError):
            data.create("/foo", "uint16_8")
        with assert_raises(ValueError):
            data.create("foo/bar", "uint16_8")
        data.create("/foo/bar", "uint16_8")
        data.create(u"/foo/baz/quux", "float64_16")
        with assert_raises(ValueError):
            data.create("/foo/bar/baz", "uint16_8")
        with assert_raises(ValueError):
            data.create("/foo/baz", "float64_16")

        # get node -- see if caching works
        nodes = []
        for i in range(5000):
            nodes.append(data.getnode("/foo/bar"))
            nodes.append(data.getnode("/foo/baz/quux"))
        del nodes

        def get_node_slice(key):
            if isinstance(key, slice):
                return [ node.get_data(x, x+1) for x in
```

```
                    xrange(*key.indices(node.nrows)) ]
        return node.get_data(key, key+1)

    # Test node
    node = data.getnode("/foo/bar")
    with assert_raises(IndexError):
        x = get_node_slice(0)
    with assert_raises(IndexError):
        x = node[0] # timestamp
    raw = []
    for i in range(1000):
        raw.append("%d 1 2 3 4 5 6 7 8\n" % (10000 + i))
    node.append_data("".join(raw[0:1]), 0, 50000)
    node.append_data("".join(raw[1:100]), 0, 50000)
    node.append_data("".join(raw[100:]), 0, 50000)

    misc_slices = [ 0, 100, slice(None), slice(0), slice(10),
                    slice(5,10), slice(3,None), slice(3,-3),
                    slice(20,10), slice(200,100,-1), slice(None,0,-1),
                    slice(100,500,5) ]

    # Extract slices
    for s in misc_slices:
        eq_(get_node_slice(s), raw[s])

    # Extract misc slices while appending, to make sure the
    # data isn't being added in the middle of the file
    for s in [2, slice(1,5), 2, slice(1,5)]:
        node.append_data("0 0 0 0 0 0 0 0\n", 0, 50000)
        raw.append("0 0 0 0 0 0 0 0\n")
        eq_(get_node_slice(s), raw[s])

    # Get some coverage of remove; remove is more fully tested
    # in cmdline
    with assert_raises(IndexError):
        node.remove(9999,9998)

    # close, reopen
    # reopen
    data.close()
    if not size or not files:
        data = BulkData(db)
    else:
        data = BulkData(db, file_size = size, files_per_dir = files)
    node = data.getnode("/foo/bar")

    # Extract slices
    for s in misc_slices:
        eq_(get_node_slice(s), raw[s])

    # destroy
    with assert_raises(ValueError):
        data.destroy("/foo")
    with assert_raises(ValueError):
        data.destroy("/foo/baz")
    with assert_raises(ValueError):
        data.destroy("/foo/qwerty")
    data.destroy("/foo/baz/quux")
    data.destroy("/foo/bar")

    # close
    data.close()
```

Listing C-13: `tests/test_nilmdb.py`: Test the main NilmDB server and class. This includes tests for specific HTTP behaviors.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/test_nilmdb.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
import nilmdb.server

from nose.tools import *
from nose.tools import assert_raises
import distutils.version
import simplejson as json
import itertools
import os
import sys
import threading
import urllib2
from urllib2 import urlopen, HTTPError
import cStringIO
import time
import requests

from nilmdb.utils import serializer_proxy

testdb = "tests/testdb"

#@atexit.register
#def cleanup():
#    os.unlink(testdb)

from testutil.helpers import *

class Test00Nilmdb(object):  # named 00 so it runs first
    def test_NilmDB(self):
        recursive_unlink(testdb)

        db = nilmdb.server.NilmDB(testdb)
        db.close()
        db = nilmdb.server.NilmDB(testdb)
        db.close()

        # test timer, just to get coverage
        capture = cStringIO.StringIO()
        old = sys.stdout
        sys.stdout = capture
        with nilmdb.utils.Timer("test"):
            time.sleep(0.01)
        sys.stdout = old
        in_("test: ", capture.getvalue())

    def test_stream(self):
        db = nilmdb.server.NilmDB(testdb)
        eq_(db.stream_list(), [])

        # Bad path
        with assert_raises(ValueError):
            db.stream_create("foo/bar/baz", "float32_8")
        with assert_raises(ValueError):
            db.stream_create("/foo", "float32_8")
        # Bad layout type
        with assert_raises(ValueError):
            db.stream_create("/newton/prep", "NoSuchLayout")
        db.stream_create("/newton/prep", "float32_8")
        db.stream_create("/newton/raw", "uint16_6")
        db.stream_create("/newton/zzz/rawnotch", "uint16_9")
```

```
60
61          # Verify we got 3 streams
62          eq_(db.stream_list(), [ ["/newton/prep", "float32_8"],
63                                  ["/newton/raw", "uint16_6"],
64                                  ["/newton/zzz/rawnotch", "uint16_9"]
65                                  ])
66          # Match just one type or one path
67          eq_(db.stream_list(layout="uint16_6"), [ ["/newton/raw", "uint16_6"] ])
68          eq_(db.stream_list(path="/newton/raw"), [ ["/newton/raw", "uint16_6"] ])
69
70          # Verify that columns were made right (pytables specific)
71          if "h5file" in db.data.__dict__:
72              h5file = db.data.h5file
73              eq_(len(h5file.getNode("/newton/prep").cols), 9)
74              eq_(len(h5file.getNode("/newton/raw").cols), 7)
75              eq_(len(h5file.getNode("/newton/zzz/rawnotch").cols), 10)
76              assert(not h5file.getNode("/newton/prep").colindexed["timestamp"])
77              assert(not h5file.getNode("/newton/prep").colindexed["c1"])
78
79          # Set / get metadata
80          eq_(db.stream_get_metadata("/newton/prep"), {})
81          eq_(db.stream_get_metadata("/newton/raw"), {})
82          meta1 = { "description": "The Data",
83                    "v_scale": "1.234" }
84          meta2 = { "description": "The Data" }
85          meta3 = { "v_scale": "1.234" }
86          db.stream_set_metadata("/newton/prep", meta1)
87          db.stream_update_metadata("/newton/prep", {})
88          db.stream_update_metadata("/newton/raw", meta2)
89          db.stream_update_metadata("/newton/raw", meta3)
90          eq_(db.stream_get_metadata("/newton/prep"), meta1)
91          eq_(db.stream_get_metadata("/newton/raw"), meta1)
92
93          # fill in some misc. test coverage
94          with assert_raises(nilmdb.server.NilmDBError):
95              db.stream_remove("/newton/prep", 0, 0)
96          with assert_raises(nilmdb.server.NilmDBError):
97              db.stream_remove("/newton/prep", 1, 0)
98          db.stream_remove("/newton/prep", 0, 1)
99
100         with assert_raises(nilmdb.server.NilmDBError):
101             db.stream_extract("/newton/prep", count = True, binary = True)
102
103         db.close()
104
105 class TestBlockingServer(object):
106     def setUp(self):
107         self.db = serializer_proxy(nilmdb.server.NilmDB)(testdb)
108
109     def tearDown(self):
110         self.db.close()
111
112     def test_blocking_server(self):
113         # Server should fail if the database doesn't have a "_thread_safe"
114         # property.
115         with assert_raises(KeyError):
116             nilmdb.server.Server(object())
117
118         # Start web app on a custom port
119         self.server = nilmdb.server.Server(self.db, host = "127.0.0.1",
120                                            port = 32180, stoppable = True)
121
122         # Run it
123         event = threading.Event()
124         def run_server():
125             self.server.start(blocking = True, event = event)
126         thread = threading.Thread(target = run_server)
```

```
127         thread.start()
128         if not event.wait(timeout = 10):
129             raise AssertionError("server didn't start in 10 seconds")
130 
131         # Send request to exit.
132         req = urlopen("http://127.0.0.1:32180/exit/", timeout = 1)
133 
134         # Wait for it
135         thread.join()
136 
137 def geturl(path):
138     req = urlopen("http://127.0.0.1:32180" + path, timeout = 10)
139     return req.read()
140 
141 def getjson(path):
142     return json.loads(geturl(path))
143 
144 class TestServer(object):
145 
146     def setUp(self):
147         # Start web app on a custom port
148         self.db = serializer_proxy(nilmdb.server.NilmDB)(testdb)
149         self.server = nilmdb.server.Server(self.db, host = "127.0.0.1",
150                                            port = 32180, stoppable = False)
151         self.server.start(blocking = False)
152 
153     def tearDown(self):
154         # Close web app
155         self.server.stop()
156         self.db.close()
157 
158     def test_server(self):
159         # Make sure we can't force an exit, and test other 404 errors
160         for url in [ "/exit", "/favicon.ico" ]:
161             with assert_raises(HTTPError) as e:
162                 geturl(url)
163             eq_(e.exception.code, 404)
164 
165         # Root page
166         in_("This is NilmDB", geturl("/"))
167 
168         # Check version
169         eq_(distutils.version.LooseVersion(getjson("/version")),
170             distutils.version.LooseVersion(nilmdb.__version__))
171 
172     def test_stream_list(self):
173         # Known streams that got populated by an earlier test (test_nilmdb)
174         streams = getjson("/stream/list")
175 
176         eq_(streams, [
177             ['/newton/prep', 'float32_8'],
178             ['/newton/raw', 'uint16_6'],
179             ['/newton/zzz/rawnotch', 'uint16_9'],
180             ])
181 
182         streams = getjson("/stream/list?layout=uint16_6")
183         eq_(streams, [['/newton/raw', 'uint16_6']])
184 
185         streams = getjson("/stream/list?layout=NoSuchLayout")
186         eq_(streams, [])
187 
188 
189     def test_stream_metadata(self):
190         with assert_raises(HTTPError) as e:
191             getjson("/stream/get_metadata?path=foo")
192         eq_(e.exception.code, 404)
193 
```

```
194         data = getjson("/stream/get_metadata?path=/newton/prep")
195         eq_(data, {'description': 'The Data', 'v_scale': '1.234'})
196 
197         data = getjson("/stream/get_metadata?path=/newton/prep"
198                        "&key=v_scale")
199         eq_(data, {'v_scale': '1.234'})
200 
201         data = getjson("/stream/get_metadata?path=/newton/prep"
202                        "&key=v_scale&key=description")
203         eq_(data, {'description': 'The Data', 'v_scale': '1.234'})
204 
205         data = getjson("/stream/get_metadata?path=/newton/prep"
206                        "&key=v_scale&key=foo")
207         eq_(data, {'foo': None, 'v_scale': '1.234'})
208 
209         data = getjson("/stream/get_metadata?path=/newton/prep"
210                        "&key=foo")
211         eq_(data, {'foo': None})
212 
213     def test_cors_headers(self):
214         # Test that CORS headers are being set correctly
215 
216         # Normal GET should send simple response
217         url = "http://127.0.0.1:32180/stream/list"
218         r = requests.get(url, headers = { "Origin": "http://google.com/" })
219         eq_(r.status_code, 200)
220         if "access-control-allow-origin" not in r.headers:
221             raise AssertionError("No Access-Control-Allow-Origin (CORS) "
222                                  "header in response:\n", r.headers)
223         eq_(r.headers["access-control-allow-origin"], "http://google.com/")
224 
225         # OPTIONS without CORS preflight headers should result in 405
226         r = requests.options(url, headers = {
227             "Origin": "http://google.com/",
228             })
229         eq_(r.status_code, 405)
230 
231         # OPTIONS with preflight headers should give preflight response
232         r = requests.options(url, headers = {
233             "Origin": "http://google.com/",
234             "Access-Control-Request-Method": "POST",
235             "Access-Control-Request-Headers": "X-Custom",
236             })
237         eq_(r.status_code, 200)
238         if "access-control-allow-origin" not in r.headers:
239             raise AssertionError("No Access-Control-Allow-Origin (CORS) "
240                                  "header in response:\n", r.headers)
241         eq_(r.headers["access-control-allow-methods"], "GET, HEAD")
242         eq_(r.headers["access-control-allow-headers"], "X-Custom")
243 
244     def test_post_bodies(self):
245         # Test JSON post bodies
246         r = requests.post("http://127.0.0.1:32180/stream/set_metadata",
247                           headers = { "Content-Type": "application/json" },
248                           data = '{"hello": 1}')
249         eq_(r.status_code, 404) # wrong parameters
250 
251         r = requests.post("http://127.0.0.1:32180/stream/set_metadata",
252                           headers = { "Content-Type": "application/json" },
253                           data = '["hello"]')
254         eq_(r.status_code, 415) # not a dict
255 
256         r = requests.post("http://127.0.0.1:32180/stream/set_metadata",
257                           headers = { "Content-Type": "application/json" },
258                           data = '[hello]')
259         eq_(r.status_code, 400) # badly formatted JSON
```

Listing C-14: `tests/test_client.py`: Test client library and context managers.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_client.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# -*- coding: utf-8 -*-

import nilmdb.server
import nilmdb.client

from nilmdb.utils.printf import *
from nilmdb.utils import timestamper
from nilmdb.client import ClientError, ServerError
from nilmdb.utils import datetime_tz

from nose.plugins.skip import SkipTest
from nose.tools import *
from nose.tools import assert_raises
import itertools
import distutils.version
import os
import sys
import threading
import cStringIO
import simplejson as json
import unittest
import warnings
import resource
import time
import re
import struct

from testutil.helpers import *

testdb = "tests/client-testdb"
testurl = "http://localhost:32180/"

def setup_module():
    global test_server, test_db
    # Clear out DB
    recursive_unlink(testdb)

    # Start web app on a custom port
    test_db = nilmdb.utils.serializer_proxy(nilmdb.server.NilmDB)(testdb)
    test_server = nilmdb.server.Server(test_db, host = "127.0.0.1",
                                       port = 32180, stoppable = False,
                                       fast_shutdown = True,
                                       force_traceback = True)
    test_server.start(blocking = False)

def teardown_module():
    global test_server, test_db
    # Close web app
    test_server.stop()
    test_db.close()

class TestClient(object):

    def test_client_01_basic(self):
        # Test a fake host
        client = nilmdb.client.Client(url = "http://localhost:1/")
        with assert_raises(nilmdb.client.ServerError):
            client.version()
        client.close()
```

```
60
61          # Then a fake URL on a real host
62          client = nilmdb.client.Client(url = "http://localhost:32180/fake/")
63          with assert_raises(nilmdb.client.ClientError):
64              client.version()
65          client.close()
66
67          # Now a real URL with no http:// prefix
68          client = nilmdb.client.Client(url = "localhost:32180")
69          version = client.version()
70          client.close()
71
72          # Now use the real URL
73          client = nilmdb.client.Client(url = testurl)
74          version = client.version()
75          eq_(distutils.version.LooseVersion(version),
76              distutils.version.LooseVersion(test_server.version))
77
78          # Bad URLs should give 404, not 500
79          with assert_raises(ClientError):
80              client.http.get("/stream/create")
81          client.close()
82
83      def test_client_02_createlist(self):
84          # Basic stream tests, like those in test_nilmdb:test_stream
85          client = nilmdb.client.Client(url = testurl)
86
87          # Database starts empty
88          eq_(client.stream_list(), [])
89
90          # Bad path
91          with assert_raises(ClientError):
92              client.stream_create("foo/bar/baz", "float32_8")
93          with assert_raises(ClientError):
94              client.stream_create("/foo", "float32_8")
95          # Bad layout type
96          with assert_raises(ClientError):
97              client.stream_create("/newton/prep", "NoSuchLayout")
98
99          # Bad method types
100         with assert_raises(ClientError):
101             client.http.put("/stream/list","")
102         # Try a bunch of times to make sure the request body is getting consumed
103         for x in range(10):
104             with assert_raises(ClientError):
105                 client.http.post("/stream/list")
106         client = nilmdb.client.Client(url = testurl)
107
108         # Create four streams
109         client.stream_create("/newton/prep", "float32_8")
110         client.stream_create("/newton/raw", "uint16_6")
111         client.stream_create("/newton/zzz/rawnotch2", "uint16_9")
112         client.stream_create("/newton/zzz/rawnotch11", "uint16_9")
113
114         # Verify we got 4 streams in the right order
115         eq_(client.stream_list(), [ ["/newton/prep", "float32_8"],
116                                     ["/newton/raw", "uint16_6"],
117                                     ["/newton/zzz/rawnotch2", "uint16_9"],
118                                     ["/newton/zzz/rawnotch11", "uint16_9"]
119                                     ])
120
121         # Match just one type or one path
122         eq_(client.stream_list(layout="uint16_6"),
123             [ ["/newton/raw", "uint16_6"] ])
124         eq_(client.stream_list(path="/newton/raw"),
125             [ ["/newton/raw", "uint16_6"] ])
126
```

```
127         # Try messing with resource limits to trigger errors and get
128         # more coverage.  Here, make it so we can only create files 1
129         # byte in size, which will trigger an IOError in the server when
130         # we create a table.
131         limit = resource.getrlimit(resource.RLIMIT_FSIZE)
132         resource.setrlimit(resource.RLIMIT_FSIZE, (1, limit[1]))
133         with assert_raises(ServerError) as e:
134             client.stream_create("/newton/hello", "uint16_6")
135         resource.setrlimit(resource.RLIMIT_FSIZE, limit)
136 
137         client.close()
138 
139     def test_client_03_metadata(self):
140         client = nilmdb.client.Client(url = testurl)
141 
142         # Set / get metadata
143         eq_(client.stream_get_metadata("/newton/prep"), {})
144         eq_(client.stream_get_metadata("/newton/raw"), {})
145         meta1 = { "description": "The Data",
146                   "v_scale": "1.234" }
147         meta2 = { "description": "The Data" }
148         meta3 = { "v_scale": "1.234" }
149         client.stream_set_metadata("/newton/prep", meta1)
150         client.stream_update_metadata("/newton/prep", {})
151         client.stream_update_metadata("/newton/raw", meta2)
152         client.stream_update_metadata("/newton/raw", meta3)
153         eq_(client.stream_get_metadata("/newton/prep"), meta1)
154         eq_(client.stream_get_metadata("/newton/raw"), meta1)
155         eq_(client.stream_get_metadata("/newton/raw",
156                                        [ "description" ] ), meta2)
157         eq_(client.stream_get_metadata("/newton/raw",
158                                        [ "description", "v_scale" ] ), meta1)
159 
160         # missing key
161         eq_(client.stream_get_metadata("/newton/raw", "descr"),
162             { "descr": None })
163         eq_(client.stream_get_metadata("/newton/raw", [ "descr" ]),
164             { "descr": None })
165 
166         # test wrong types (list instead of dict)
167         with assert_raises(ClientError):
168             client.stream_set_metadata("/newton/prep", [1,2,3])
169         with assert_raises(ClientError):
170             client.stream_update_metadata("/newton/prep", [1,2,3])
171 
172         # test wrong types (dict of non-strings)
173         # numbers are OK; they'll get converted to strings
174         client.stream_set_metadata("/newton/prep", { "hello": 1234 })
175         # anything else is not
176         with assert_raises(ClientError):
177             client.stream_set_metadata("/newton/prep", { "world": { 1: 2 } })
178         with assert_raises(ClientError):
179             client.stream_set_metadata("/newton/prep", { "world": [ 1, 2 ] })
180 
181         client.close()
182 
183     def test_client_04_insert(self):
184         client = nilmdb.client.Client(url = testurl)
185 
186         # Limit _max_data to 1 MB, since our test file is 1.5 MB
187         old_max_data = nilmdb.client.client.StreamInserter._max_data
188         nilmdb.client.client.StreamInserter._max_data = 1 * 1024 * 1024
189 
190         datetime_tz.localtz_set("America/New_York")
191 
192         testfile = "tests/data/prep-20120323T1000"
193         start = nilmdb.utils.time.parse_time("20120323T1000")
```

```
        rate = 120

        # First try a nonexistent path
        data = timestamper.TimestamperRate(testfile, start, 120)
        with assert_raises(ClientError) as e:
            result = client.stream_insert("/newton/no-such-path", data)
        in_("404 Not Found", str(e.exception))

        # Now try reversed timestamps
        data = timestamper.TimestamperRate(testfile, start, 120)
        data = reversed(list(data))
        with assert_raises(ClientError) as e:
            result = client.stream_insert("/newton/prep", data)
        in_("400 Bad Request", str(e.exception))
        in2_("timestamp is not monotonically increasing",
             "start must precede end", str(e.exception))

        # Now try empty data (no server request made)
        empty = cStringIO.StringIO("")
        data = timestamper.TimestamperRate(empty, start, 120)
        result = client.stream_insert("/newton/prep", data)
        eq_(result, None)

        # It's OK to insert an empty interval
        client.http.put("stream/insert", "", { "path": "/newton/prep",
                                               "start": 1, "end": 2 })
        eq_(list(client.stream_intervals("/newton/prep")), [[1, 2]])
        client.stream_remove("/newton/prep")
        eq_(list(client.stream_intervals("/newton/prep")), [])

        # Timestamps can be negative too
        client.http.put("stream/insert", "", { "path": "/newton/prep",
                                               "start": -2, "end": -1 })
        eq_(list(client.stream_intervals("/newton/prep")), [[-2, -1]])
        client.stream_remove("/newton/prep")
        eq_(list(client.stream_intervals("/newton/prep")), [])

        # Intervals that end at zero shouldn't be any different
        client.http.put("stream/insert", "", { "path": "/newton/prep",
                                               "start": -1, "end": 0 })
        eq_(list(client.stream_intervals("/newton/prep")), [[-1, 0]])
        client.stream_remove("/newton/prep")
        eq_(list(client.stream_intervals("/newton/prep")), [])

        # Try forcing a server request with equal start and end
        with assert_raises(ClientError) as e:
            client.http.put("stream/insert", "", { "path": "/newton/prep",
                                                   "start": 0, "end": 0 })
        in_("400 Bad Request", str(e.exception))
        in_("start must precede end", str(e.exception))

        # Invalid times in HTTP request
        with assert_raises(ClientError) as e:
            client.http.put("stream/insert", "", { "path": "/newton/prep",
                                                   "start": "asdf", "end": 0 })
        in_("400 Bad Request", str(e.exception))
        in_("invalid start", str(e.exception))

        with assert_raises(ClientError) as e:
            client.http.put("stream/insert", "", { "path": "/newton/prep",
                                                   "start": 0, "end": "asdf" })
        in_("400 Bad Request", str(e.exception))
        in_("invalid end", str(e.exception))

        # Good content type
        with assert_raises(ClientError) as e:
            client.http.put("stream/insert", "",
```

```
                           { "path": "xxxx", "start": 0, "end": 1,
                             "binary": 1 },
                           binary = True)
        in_("No such stream", str(e.exception))

        # Bad content type
        with assert_raises(ClientError) as e:
            client.http.put("stream/insert", "",
                           { "path": "xxxx", "start": 0, "end": 1,
                             "binary": 1 },
                           binary = False)
        in_("Content type must be application/octet-stream", str(e.exception))

        # Specify start/end (starts too late)
        data = timestamper.TimestamperRate(testfile, start, 120)
        with assert_raises(ClientError) as e:
            result = client.stream_insert("/newton/prep", data,
                                          start + 5000000, start + 120000000)
        in_("400 Bad Request", str(e.exception))
        in_("Data timestamp 1332511200000000 < start time 1332511205000000",
            str(e.exception))

        # Specify start/end (ends too early)
        data = timestamper.TimestamperRate(testfile, start, 120)
        with assert_raises(ClientError) as e:
            result = client.stream_insert("/newton/prep", data,
                                          start, start + 1000000)
        in_("400 Bad Request", str(e.exception))
        # Client chunks the input, so the exact timestamp here might change
        # if the chunk positions change.
        assert(re.search("Data timestamp 13325[0-9]+ "
                         ">= end time 1332511201000000", str(e.exception))
               is not None)

        # Now do the real load
        data = timestamper.TimestamperRate(testfile, start, 120)
        result = client.stream_insert("/newton/prep", data,
                                      start, start + 119999777)

        # Verify the intervals.  Should be just one, even if the data
        # was inserted in chunks, due to nilmdb interval concatenation.
        intervals = list(client.stream_intervals("/newton/prep"))
        eq_(intervals, [[start, start + 119999777]])

        # Try some overlapping data -- just insert it again
        data = timestamper.TimestamperRate(testfile, start, 120)
        with assert_raises(ClientError) as e:
            result = client.stream_insert("/newton/prep", data)
        in_("400 Bad Request", str(e.exception))
        in_("verlap", str(e.exception))

        nilmdb.client.client.StreamInserter._max_data = old_max_data
        client.close()

    def test_client_05_extractremove(self):
        # Misc tests for extract and remove.  Most of them are in test_cmdline.
        client = nilmdb.client.Client(url = testurl)

        for x in client.stream_extract("/newton/prep",
                                       999123000000, 999124000000):
            raise AssertionError("shouldn't be any data for this request")

        with assert_raises(ClientError) as e:
            client.stream_remove("/newton/prep", 123000000, 120000000)

        # Test count
        eq_(client.stream_count("/newton/prep"), 14400)
```

```
328
329         # Test binary output
330         with assert_raises(ClientError) as e:
331             list(client.stream_extract("/newton/prep",
332                                        markup = True, binary = True))
333         with assert_raises(ClientError) as e:
334             list(client.stream_extract("/newton/prep",
335                                        count = True, binary = True))
336         data = "".join(client.stream_extract("/newton/prep", binary = True))
337         # Quick check using struct
338         unpacker = struct.Struct("<qffffffff")
339         out = []
340         for i in range(14400):
341             out.append(unpacker.unpack_from(data, i * unpacker.size))
342         eq_(out[0], (1332511200000000, 266568.0, 224029.0, 5161.39990234375,
343                      2525.169921875, 8350.83984375, 3724.699951171875,
344                      1355.3399658203125, 2039.0))
345
346         # Just get some coverage
347         with assert_raises(ClientError) as e:
348             client.http.post("/stream/remove", { "path": "/none" })
349
350         client.close()
351
352     def test_client_06_generators(self):
353         # A lot of the client functionality is already tested by test_cmdline,
354         # but this gets a bit more coverage that cmdline misses.
355         client = nilmdb.client.Client(url = testurl)
356
357         # Trigger a client error in generator
358         start = nilmdb.utils.time.parse_time("20120323T2000")
359         end = nilmdb.utils.time.parse_time("20120323T1000")
360         for function in [ client.stream_intervals, client.stream_extract ]:
361             with assert_raises(ClientError) as e:
362                 function("/newton/prep", start, end).next()
363             in_("400 Bad Request", str(e.exception))
364             in_("start must precede end", str(e.exception))
365
366         # Trigger a curl error in generator
367         with assert_raises(ServerError) as e:
368             client.http.get_gen("http://nosuchurl.example.com./").next()
369
370         # Check 404 for missing streams
371         for function in [ client.stream_intervals, client.stream_extract ]:
372             with assert_raises(ClientError) as e:
373                 function("/no/such/stream").next()
374             in_("404 Not Found", str(e.exception))
375             in_("No such stream", str(e.exception))
376
377         client.close()
378
379     def test_client_07_headers(self):
380         # Make sure that /stream/intervals and /stream/extract
381         # properly return streaming, chunked, text/plain response.
382         # Pokes around in client.http internals a bit to look at the
383         # response headers.
384
385         client = nilmdb.client.Client(url = testurl)
386         http = client.http
387
388         # Use a warning rather than returning a test failure for the
389         # transfer-encoding, so that we can still disable chunked
390         # responses for debugging.
391
392         def headers():
393             h = ""
394             for (k, v) in http._last_response.headers.items():
```

```
                h += k + ": " + v + "\n"
            return h.lower()

        # Intervals
        x = http.get("stream/intervals", { "path": "/newton/prep" })
        if "transfer-encoding: chunked" not in headers():
            warnings.warn("Non-chunked HTTP response for /stream/intervals")
        if "content-type: application/x-json-stream" not in headers():
            raise AssertionError("/stream/intervals content type "
                                 "is not application/x-json-stream:\n" +
                                 headers())

        # Extract
        x = http.get("stream/extract", { "path": "/newton/prep",
                                         "start": "123", "end": "124" })
        if "transfer-encoding: chunked" not in headers():
            warnings.warn("Non-chunked HTTP response for /stream/extract")
        if "content-type: text/plain;charset=utf-8" not in headers():
            raise AssertionError("/stream/extract is not text/plain:\n" +
                                 headers())

        x = http.get("stream/extract", { "path": "/newton/prep",
                                         "start": "123", "end": "124",
                                         "binary": "1" })
        if "transfer-encoding: chunked" not in headers():
            warnings.warn("Non-chunked HTTP response for /stream/extract")
        if "content-type: application/octet-stream" not in headers():
            raise AssertionError("/stream/extract is not binary:\n" +
                                 headers())

        # Make sure a binary of "0" is really off
        x = http.get("stream/extract", { "path": "/newton/prep",
                                         "start": "123", "end": "124",
                                         "binary": "0" })
        if "content-type: application/octet-stream" in headers():
                    raise AssertionError("/stream/extract is not text:\n" +
                                         headers())

        # Invalid parameters
        with assert_raises(ClientError) as e:
            x = http.get("stream/extract", { "path": "/newton/prep",
                                             "start": "123", "end": "124",
                                             "binary": "asdfasfd" })
        in_("can't parse parameter", str(e.exception))

        client.close()

    def test_client_08_unicode(self):
        # Try both with and without posting JSON
        for post_json in (False, True):
            # Basic Unicode tests
            client = nilmdb.client.Client(url = testurl, post_json = post_json)

            # Delete streams that exist
            for stream in client.stream_list():
                client.stream_remove(stream[0])
                client.stream_destroy(stream[0])

            # Database is empty
            eq_(client.stream_list(), [])

            # Create Unicode stream, match it
            raw = [ u"/düsseldorf/raw", u"uint16_6" ]
            prep = [ u"/düsseldorf/prep", u"uint16_6" ]
            client.stream_create(*raw)
            eq_(client.stream_list(), [raw])
            eq_(client.stream_list(layout=raw[1]), [raw])
```

```
462             eq_(client.stream_list(path=raw[0]), [raw])
463             client.stream_create(*prep)
464             eq_(client.stream_list(), [prep, raw])
465
466             # Set / get metadata with Unicode keys and values
467             eq_(client.stream_get_metadata(raw[0]), {})
468             eq_(client.stream_get_metadata(prep[0]), {})
469             meta1 = { u"alpha": u"α",
470                       u"β": u"beta" }
471             meta2 = { u"alpha": u"α" }
472             meta3 = { u"β": u"beta" }
473             client.stream_set_metadata(prep[0], meta1)
474             client.stream_update_metadata(prep[0], {})
475             client.stream_update_metadata(raw[0], meta2)
476             client.stream_update_metadata(raw[0], meta3)
477             eq_(client.stream_get_metadata(prep[0]), meta1)
478             eq_(client.stream_get_metadata(raw[0]), meta1)
479             eq_(client.stream_get_metadata(raw[0], [ "alpha" ]), meta2)
480             eq_(client.stream_get_metadata(raw[0], [ "alpha", "β" ]), meta1)
481
482             client.close()
483
484     def test_client_09_closing(self):
485         # Make sure we actually close sockets correctly.  New
486         # connections will block for a while if they're not, since the
487         # server will stop accepting new connections.
488         for test in [1, 2]:
489             start = time.time()
490             for i in range(50):
491                 if time.time() - start > 15:
492                     raise AssertionError("Connections seem to be blocking... "
493                                          "probably not closing properly.")
494                 if test == 1:
495                     # explicit close
496                     client = nilmdb.client.Client(url = testurl)
497                     with assert_raises(ClientError) as e:
498                         client.stream_remove("/newton/prep", 123, 120)
499                     client.close() # remove this to see the failure
500                 elif test == 2:
501                     # use the context manager
502                     with nilmdb.client.Client(url = testurl) as c:
503                         with assert_raises(ClientError) as e:
504                             c.stream_remove("/newton/prep", 123, 120)
505
506     def test_client_10_context(self):
507         # Test using the client's stream insertion context manager to
508         # insert data.
509         client = nilmdb.client.Client(testurl)
510
511         client.stream_create("/context/test", "uint16_1")
512         with client.stream_insert_context("/context/test") as ctx:
513             # override _max_data to trigger frequent server updates
514             ctx._max_data = 15
515
516             ctx.insert("1000 1\n")
517
518             ctx.insert("1010 ")
519             ctx.insert("1\n1020 1")
520             ctx.insert("")
521             ctx.insert("\n1030 1\n")
522
523             ctx.insert("1040 1\n")
524             ctx.insert("# hello\n")
525             ctx.insert("   # hello\n")
526             ctx.insert("  1050 1\n")
527             ctx.finalize()
528
```

```
            ctx.insert("1070 1\n")
            ctx.update_end(1080)
            ctx.finalize()
            ctx.update_start(1090)
            ctx.insert("1100 1\n")
            ctx.insert("1110 1\n")
            ctx.send()
            ctx.insert("1120 1\n")
            ctx.insert("1130 1\n")
            ctx.insert("1140 1\n")
            ctx.update_end(1160)
            ctx.insert("1150 1\n")
            ctx.update_end(1170)
            ctx.insert("1160 1\n")
            ctx.update_end(1180)
            ctx.insert("1170 1" +
                       " # this is super long" * 100 +
                       "\n")
            ctx.finalize()
            ctx.insert("# this is super long" * 100)

        with assert_raises(ClientError):
            with client.stream_insert_context("/context/test",
                                              1000, 2000) as ctx:
                ctx.insert("1180 1\n")

        with assert_raises(ClientError):
            with client.stream_insert_context("/context/test",
                                              2000, 3000) as ctx:
                ctx.insert("1180 1\n")

        with assert_raises(ClientError):
            with client.stream_insert_context("/context/test") as ctx:
                ctx.insert("bogus data\n")

        with client.stream_insert_context("/context/test", 2000, 3000) as ctx:
            # make sure our override wasn't permanent
            ne_(ctx._max_data, 15)
            ctx.insert("2250 1\n")
            ctx.finalize()

        with assert_raises(ClientError):
            with client.stream_insert_context("/context/test",
                                              3000, 4000) as ctx:
                ctx.insert("3010 1\n")
                ctx.insert("3020 2\n")
                ctx.insert("3030 3\n")
                ctx.insert("3040 4\n")
                ctx.insert("3040 4\n") # non-monotonic after a few lines
                ctx.finalize()

        eq_(list(client.stream_intervals("/context/test")),
            [ [ 1000, 1051 ],
              [ 1070, 1080 ],
              [ 1090, 1180 ],
              [ 2000, 3000 ] ])

        # destroy stream (try without removing data first)
        with assert_raises(ClientError):
            client.stream_destroy("/context/test")
        client.stream_remove("/context/test")
        client.stream_destroy("/context/test")
        client.close()

    def test_client_11_emptyintervals(self):
        # Empty intervals are ok!  If recording detection events
        # by inserting rows into the database, we want to be able to
```

```
        # have an interval where no events occurred.  Test them here.
        client = nilmdb.client.Client(testurl)
        client.stream_create("/empty/test", "uint16_1")

        def info():
            result = []
            for interval in list(client.stream_intervals("/empty/test")):
                result.append((client.stream_count("/empty/test", *interval),
                               interval))
            return result

        eq_(info(), [])

        # Insert a region with just a few points
        with client.stream_insert_context("/empty/test") as ctx:
            ctx.update_start(100)
            ctx.insert("140 1\n")
            ctx.insert("150 1\n")
            ctx.insert("160 1\n")
            ctx.update_end(200)
            ctx.finalize()

        eq_(info(), [(3, [100, 200])])

        # Delete chunk, which will leave one data point and two intervals
        client.stream_remove("/empty/test", 145, 175)
        eq_(info(), [(1, [100, 145]),
                     (0, [175, 200])])

        # Try also creating a completely empty interval from scratch,
        # in a few different ways.
        client.stream_insert("/empty/test", "", 300, 350)
        client.stream_insert("/empty/test", [], 400, 450)
        with client.stream_insert_context("/empty/test", 500, 550):
            pass

        # If enough timestamps aren't provided, empty streams won't be created.
        client.stream_insert("/empty/test", [])
        with client.stream_insert_context("/empty/test"):
            pass
        client.stream_insert("/empty/test", [], start = 600)
        with client.stream_insert_context("/empty/test", start = 700):
            pass
        client.stream_insert("/empty/test", [], end = 850)
        with client.stream_insert_context("/empty/test", end = 950):
            pass

        # Equal start and end is OK as long as there's no data
        with client.stream_insert_context("/empty/test", start=9, end=9):
            pass

        # Try various things that might cause problems
        with client.stream_insert_context("/empty/test", 1000, 1050) as ctx:
            ctx.finalize() # inserts [1000, 1050]
            ctx.finalize() # nothing
            ctx.finalize() # nothing
            ctx.insert("1100 1\n")
            ctx.finalize() # inserts [1100, 1101]
            ctx.update_start(1199)
            ctx.insert("1200 1\n")
            ctx.update_end(1250)
            ctx.finalize() # inserts [1199, 1250]
            ctx.update_start(1299)
            ctx.finalize() # nothing
            ctx.update_end(1350)
            ctx.finalize() # nothing
            ctx.update_start(1400)
```

```
663             ctx.insert("# nothing!\n")
664             ctx.update_end(1450)
665             ctx.finalize()
666             ctx.update_start(1500)
667             ctx.insert("# nothing!")
668             ctx.update_end(1550)
669             ctx.finalize()
670             ctx.insert("# nothing!\n" * 10)
671             ctx.finalize()
672             # implicit last finalize inserts [1400, 1450]
673 
674         # Check everything
675         eq_(info(), [(1, [100, 145]),
676                      (0, [175, 200]),
677                      (0, [300, 350]),
678                      (0, [400, 450]),
679                      (0, [500, 550]),
680                      (0, [1000, 1050]),
681                      (1, [1100, 1101]),
682                      (1, [1199, 1250]),
683                      (0, [1400, 1450]),
684                      (0, [1500, 1550]),
685                      ])
686 
687         # Clean up
688         client.stream_remove("/empty/test")
689         client.stream_destroy("/empty/test")
690         client.close()
691 
692     def test_client_12_persistent(self):
693         # Check that connections are persistent when they should be.
694         # This is pretty hard to test; we have to poke deep into
695         # the Requests library.
696         with nilmdb.client.Client(url = testurl) as c:
697             def connections():
698                 try:
699                     poolmanager = c.http._last_response.connection.poolmanager
700                     pool = poolmanager.pools[('http','localhost',32180)]
701                     return (pool.num_connections, pool.num_requests)
702                 except Exception:
703                     raise SkipTest("can't get connection info")
704 
705             # First request makes a connection
706             c.stream_create("/persist/test", "uint16_1")
707             eq_(connections(), (1, 1))
708 
709             # Non-generator
710             c.stream_list("/persist/test")
711             eq_(connections(), (1, 2))
712             c.stream_list("/persist/test")
713             eq_(connections(), (1, 3))
714 
715             # Generators
716             for x in c.stream_intervals("/persist/test"):
717                 pass
718             eq_(connections(), (1, 4))
719             for x in c.stream_intervals("/persist/test"):
720                 pass
721             eq_(connections(), (1, 5))
722 
723             # Clean up
724             c.stream_remove("/persist/test")
725             c.stream_destroy("/persist/test")
726             eq_(connections(), (1, 7))
727 
728     def test_client_13_timestamp_rounding(self):
729         # Test potentially bad timestamps (due to floating point
```

```
730         # roundoff etc).  The server will round floating point values
731         # to the nearest int.
732         client = nilmdb.client.Client(testurl)
733
734         client.stream_create("/rounding/test", "uint16_1")
735         with client.stream_insert_context("/rounding/test",
736                                           100000000, 200000000.1) as ctx:
737             ctx.insert("100000000.1 1\n")
738             ctx.insert("150000000.00003 1\n")
739             ctx.insert("199999999.4 1\n")
740         eq_(list(client.stream_intervals("/rounding/test")),
741             [ [ 100000000, 200000000 ] ])
742
743         with assert_raises(ClientError):
744             with client.stream_insert_context("/rounding/test",
745                                               200000000, 300000000) as ctx:
746                 ctx.insert("200000000 1\n")
747                 ctx.insert("250000000 1\n")
748                 # Server will round this and give an error on finalize()
749                 ctx.insert("299999999.99 1\n")
750
751         client.stream_remove("/rounding/test")
752         client.stream_destroy("/rounding/test")
753         client.close()
```

Listing C-15: `tests/test_numpyclient.py`: Test NumPy client library interface.

Git repository: https://git.jim.sh/jim/lees/nilmdb.git
Filename: tests/test_numpyclient.py
Revision: f5276e9fc862fb41b8777884b2c12434bafb71e4

```
1  # -*- coding: utf-8 -*-
2
3  import nilmdb.server
4  import nilmdb.client
5  import nilmdb.client.numpyclient
6
7  from nilmdb.utils.printf import *
8  from nilmdb.utils import timestamper
9  from nilmdb.client import ClientError, ServerError
10 from nilmdb.utils import datetime_tz
11
12 from nose.plugins.skip import SkipTest
13 from nose.tools import *
14 from nose.tools import assert_raises
15 import itertools
16 import distutils.version
17
18 from testutil.helpers import *
19
20 import numpy as np
21
22 testdb = "tests/numpyclient-testdb"
23 testurl = "http://localhost:32180/"
24
25 def setup_module():
26     global test_server, test_db
27     # Clear out DB
28     recursive_unlink(testdb)
29
30     # Start web app on a custom port
31     test_db = nilmdb.utils.serializer_proxy(nilmdb.server.NilmDB)(
```

```
        testdb, bulkdata_args = { "file_size" : 16384,
                                  "files_per_dir" : 3 } )

    test_server = nilmdb.server.Server(test_db, host = "127.0.0.1",
                                       port = 32180, stoppable = False,
                                       fast_shutdown = True,
                                       force_traceback = True)
    test_server.start(blocking = False)

def teardown_module():
    global test_server, test_db
    # Close web app
    test_server.stop()
    test_db.close()

class TestNumpyClient(object):

    def test_numpyclient_01_basic(self):
        # Test basic connection
        client = nilmdb.client.numpyclient.NumpyClient(url = testurl)
        version = client.version()
        eq_(distutils.version.LooseVersion(version),
            distutils.version.LooseVersion(test_server.version))

        # Verify subclassing
        assert(isinstance(client, nilmdb.client.Client))

        # Layouts
        for layout in "int8_t", "something_8", "integer_1":
            with assert_raises(ValueError):
                for x in client.stream_extract_numpy("/foo", layout=layout):
                    pass
        for layout in "int8_1", "uint8_30", "int16_20", "float64_100":
            with assert_raises(ClientError) as e:
                for x in client.stream_extract_numpy("/foo", layout=layout):
                    pass
            in_("No such stream", str(e.exception))

        with assert_raises(ClientError) as e:
            for x in client.stream_extract_numpy("/foo"):
                pass
        in_("can't get layout for path", str(e.exception))

        client.close()

    def test_numpyclient_02_extract(self):
        client = nilmdb.client.numpyclient.NumpyClient(url = testurl)

        # Insert some data as text
        client.stream_create("/newton/prep", "float32_8")
        testfile = "tests/data/prep-20120323T1000"
        start = nilmdb.utils.time.parse_time("20120323T1000")
        rate = 120
        data = timestamper.TimestamperRate(testfile, start, rate)
        result = client.stream_insert("/newton/prep", data,
                                      start, start + 119999777)

        # Extract Numpy arrays
        array = None
        pieces = 0
        for chunk in client.stream_extract_numpy("/newton/prep", maxrows=1000):
            pieces += 1
            if array is not None:
                array = np.vstack((array, chunk))
            else:
                array = chunk
        eq_(array.shape, (14400, 9))
```

```
        eq_(pieces, 15)

        # Try structured
        s = list(client.stream_extract_numpy("/newton/prep", structured = True))
        assert(np.array_equal(np.c_[s[0]['timestamp'], s[0]['data']], array))

        # Compare.  Will be close but not exact because the conversion
        # to and from ASCII was lossy.
        data = timestamper.TimestamperRate(testfile, start, rate)
        actual = np.fromstring(" ".join(data), sep=' ').reshape(14400, 9)
        assert(np.allclose(array, actual))

        client.close()

    def test_numpyclient_03_insert(self):
        client = nilmdb.client.numpyclient.NumpyClient(url = testurl)

        # Limit _max_data just to get better coverage
        old_max_data = nilmdb.client.numpyclient.StreamInserterNumpy._max_data
        nilmdb.client.numpyclient.StreamInserterNumpy._max_data = 100000

        client.stream_create("/test/1", "uint16_1")
        client.stream_insert_numpy("/test/1",
                                   np.array([[0, 1],
                                             [1, 2],
                                             [2, 3],
                                             [3, 4]]))

        # Wrong number of dimensions
        with assert_raises(ValueError) as e:
            client.stream_insert_numpy("/test/1",
                                       np.array([[[0, 1],
                                                  [1, 2]],
                                                 [[3, 4],
                                                  [4, 5]]]))
        in_("wrong number of dimensions", str(e.exception))

        # Wrong number of fields
        with assert_raises(ValueError) as e:
            client.stream_insert_numpy("/test/1",
                                       np.array([[0, 1, 2],
                                                 [1, 2, 3],
                                                 [3, 4, 5],
                                                 [4, 5, 6]]))
        in_("wrong number of fields", str(e.exception))

        # Unstructured
        client.stream_create("/test/2", "float32_8")
        client.stream_insert_numpy(
            "/test/2",
            client.stream_extract_numpy(
                "/newton/prep", structured = False, maxrows = 1000))

        # Structured, and specifying layout
        client.stream_create("/test/3", "float32_8")
        client.stream_insert_numpy(
            path = "/test/3", layout = "float32_8",
            data = client.stream_extract_numpy(
                "/newton/prep", structured = True, maxrows = 1000))

        # Structured, specifying wrong layout
        client.stream_create("/test/4", "float32_8")
        with assert_raises(ValueError) as e:
            client.stream_insert_numpy(
                "/test/4", layout = "uint16_1",
                data = client.stream_extract_numpy(
                    "/newton/prep", structured = True, maxrows = 1000))
```

```
        in_("wrong dtype", str(e.exception))

        # Unstructured, and specifying wrong layout
        client.stream_create("/test/5", "float32_8")
        with assert_raises(ClientError) as e:
            client.stream_insert_numpy(
                "/test/5", layout = "uint16_8",
                data = client.stream_extract_numpy(
                    "/newton/prep", structured = False, maxrows = 1000))
        # timestamps will be screwy here, because data will be parsed wrong
        in_("error parsing input data", str(e.exception))

        # Make sure the /newton/prep copies are identical
        a = np.vstack(client.stream_extract_numpy("/newton/prep"))
        b = np.vstack(client.stream_extract_numpy("/test/2"))
        c = np.vstack(client.stream_extract_numpy("/test/3"))
        assert(np.array_equal(a,b))
        assert(np.array_equal(a,c))

        # Make sure none of the files are greater than 16384 bytes as
        # we configured with the bulkdata_args above.
        datapath = os.path.join(testdb, "data")
        for (dirpath, dirnames, filenames) in os.walk(datapath):
            for f in filenames:
                fn = os.path.join(dirpath, f)
                size = os.path.getsize(fn)
                if size > 16384:
                    raise AssertionError(sprintf("%s is too big: %d > %d\n",
                                                 fn, size, 16384))

        nilmdb.client.numpyclient.StreamInserterNumpy._max_data = old_max_data
        client.close()

    def test_numpyclient_04_context(self):
        # Like test_client_context, but with Numpy data
        client = nilmdb.client.numpyclient.NumpyClient(testurl)

        client.stream_create("/context/test", "uint16_1")
        with client.stream_insert_numpy_context("/context/test") as ctx:
            # override _max_rows to trigger frequent server updates
            ctx._max_rows = 2
            ctx.insert([[1000, 1]])
            ctx.insert([[1010, 1], [1020, 1], [1030, 1]])
            ctx.insert([[1040, 1], [1050, 1]])
            ctx.finalize()
            ctx.insert([[1070, 1]])
            ctx.update_end(1080)
            ctx.finalize()
            ctx.update_start(1090)
            ctx.insert([[1100, 1]])
            ctx.insert([[1110, 1]])
            ctx.send()
            ctx.insert([[1120, 1], [1130, 1], [1140, 1]])
            ctx.update_end(1160)
            ctx.insert([[1150, 1]])
            ctx.update_end(1170)
            ctx.insert([[1160, 1]])
            ctx.update_end(1180)
            ctx.insert([[1170, 123456789.0]])
            ctx.finalize()
            ctx.insert(np.zeros((0,2)))

        with assert_raises(ClientError):
            with client.stream_insert_numpy_context("/context/test",
                                                    1000, 2000) as ctx:
                ctx.insert([[1180, 1]])

```

```
        with assert_raises(ClientError):
            with client.stream_insert_numpy_context("/context/test",
                                                    2000, 3000) as ctx:
                ctx._max_rows = 2
                ctx.insert([[3180, 1]])
                ctx.insert([[3181, 1]])

        with client.stream_insert_numpy_context("/context/test",
                                                2000, 3000) as ctx:
            # make sure our override wasn't permanent
            ne_(ctx._max_rows, 2)
            ctx.insert([[2250, 1]])
            ctx.finalize()

        with assert_raises(ClientError):
            with client.stream_insert_numpy_context("/context/test",
                                                    3000, 4000) as ctx:
                ctx.insert([[3010, 1]])
                ctx.insert([[3020, 2]])
                ctx.insert([[3030, 3]])
                ctx.insert([[3040, 4]])
                ctx.insert([[3040, 4]]) # non-monotonic after a few lines
                ctx.finalize()

        eq_(list(client.stream_intervals("/context/test")),
            [ [ 1000, 1051 ],
              [ 1070, 1080 ],
              [ 1090, 1180 ],
              [ 2000, 3000 ] ])

        client.stream_remove("/context/test")
        client.stream_destroy("/context/test")
        client.close()

    def test_numpyclient_05_emptyintervals(self):
        # Like test_client_emptyintervals, with insert_numpy_context
        client = nilmdb.client.numpyclient.NumpyClient(testurl)
        client.stream_create("/empty/test", "uint16_1")
        def info():
            result = []
            for interval in list(client.stream_intervals("/empty/test")):
                result.append((client.stream_count("/empty/test", *interval),
                               interval))
            return result
        eq_(info(), [])

        # Insert a region with just a few points
        with client.stream_insert_numpy_context("/empty/test") as ctx:
            ctx.update_start(100)
            ctx.insert([[140, 1]])
            ctx.insert([[150, 1]])
            ctx.insert([[160, 1]])
            ctx.update_end(200)
            ctx.finalize()
        eq_(info(), [(3, [100, 200])])

        # Delete chunk, which will leave one data point and two intervals
        client.stream_remove("/empty/test", 145, 175)
        eq_(info(), [(1, [100, 145]),
                     (0, [175, 200])])

        # Try also creating a completely empty interval from scratch,
        # in a few different ways.
        client.stream_insert("/empty/test", "", 300, 350)
        client.stream_insert("/empty/test", [], 400, 450)
        with client.stream_insert_numpy_context("/empty/test", 500, 550):
            pass
```

```
300
301         # If enough timestamps aren't provided, empty streams won't be created.
302         client.stream_insert("/empty/test", [])
303         with client.stream_insert_numpy_context("/empty/test"):
304             pass
305         client.stream_insert("/empty/test", [], start = 600)
306         with client.stream_insert_numpy_context("/empty/test", start = 700):
307             pass
308         client.stream_insert("/empty/test", [], end = 850)
309         with client.stream_insert_numpy_context("/empty/test", end = 950):
310             pass
311
312         # Equal start and end is OK as long as there's no data
313         with assert_raises(ClientError) as e:
314             with client.stream_insert_numpy_context("/empty/test",
315                                                     start=9, end=9) as ctx:
316                 ctx.insert([[9, 9]])
317                 ctx.finalize()
318         in_("have data to send, but invalid start/end times", str(e.exception))
319
320         with client.stream_insert_numpy_context("/empty/test",
321                                                 start=9, end=9) as ctx:
322             pass
323
324         # reusing a context object is bad
325         with assert_raises(Exception) as e:
326             ctx.insert([[9, 9]])
327
328         # Try various things that might cause problems
329         with client.stream_insert_numpy_context("/empty/test",
330                                                 1000, 1050) as ctx:
331             ctx.finalize() # inserts [1000, 1050]
332             ctx.finalize() # nothing
333             ctx.finalize() # nothing
334             ctx.insert([[1100, 1]])
335             ctx.finalize() # inserts [1100, 1101]
336             ctx.update_start(1199)
337             ctx.insert([[1200, 1]])
338             ctx.update_end(1250)
339             ctx.finalize() # inserts [1199, 1250]
340             ctx.update_start(1299)
341             ctx.finalize() # nothing
342             ctx.update_end(1350)
343             ctx.finalize() # nothing
344             ctx.update_start(1400)
345             ctx.insert(np.zeros((0,2)))
346             ctx.update_end(1450)
347             ctx.finalize()
348             ctx.update_start(1500)
349             ctx.insert(np.zeros((0,2)))
350             ctx.update_end(1550)
351             ctx.finalize()
352             ctx.insert(np.zeros((0,2)))
353             ctx.insert(np.zeros((0,2)))
354             ctx.insert(np.zeros((0,2)))
355             ctx.finalize()
356
357         # Check everything
358         eq_(info(), [(1, [100, 145]),
359                      (0, [175, 200]),
360                      (0, [300, 350]),
361                      (0, [400, 450]),
362                      (0, [500, 550]),
363                      (0, [1000, 1050]),
364                      (1, [1100, 1101]),
365                      (1, [1199, 1250]),
366                      (0, [1400, 1450]),
```

```
                      (0, [1500, 1550]),
                      ])

        # Clean up
        client.stream_remove("/empty/test")
        client.stream_destroy("/empty/test")
        client.close()
```

Listing C-16: `tests/test_cmdline.py`: Test `nilmtool` command line interface.

Git repository: `https://git.jim.sh/jim/lees/nilmdb.git`
Filename: `tests/test_cmdline.py`
Revision: `f5276e9fc862fb41b8777884b2c12434bafb71e4`

```
# -*- coding: utf-8 -*-

import nilmdb.server

from nilmdb.utils.printf import *
import nilmdb.cmdline
from nilmdb.utils import datetime_tz

import unittest
from nose.tools import *
from nose.tools import assert_raises
import itertools
import os
import re
import sys
import StringIO
import shlex
import warnings

from testutil.helpers import *

testdb = "tests/cmdline-testdb"

def server_start(max_results = None, max_removals = None, bulkdata_args = {}):
    global test_server, test_db
    # Start web app on a custom port
    test_db = nilmdb.utils.serializer_proxy(nilmdb.server.NilmDB)(
        testdb,
        max_results = max_results,
        max_removals = max_removals,
        bulkdata_args = bulkdata_args)
    test_server = nilmdb.server.Server(test_db, host = "127.0.0.1",
                                       port = 32180, stoppable = False,
                                       fast_shutdown = True,
                                       force_traceback = False)
    test_server.start(blocking = False)

def server_stop():
    global test_server, test_db
    # Close web app
    test_server.stop()
    test_db.close()

def setup_module():
    global test_server, test_db
    # Clear out DB
    recursive_unlink(testdb)
    server_start()

def teardown_module():
```

```
51      server_stop()
52
53  # Add an encoding property to StringIO so Python will convert Unicode
54  # properly when writing or reading.
55  class UTF8StringIO(StringIO.StringIO):
56      encoding = 'utf-8'
57
58  class TestCmdline(object):
59
60      def run(self, arg_string, infile=None, outfile=None):
61          """Run a cmdline client with the specified argument string,
62          passing the given input.  Save the output and exit code."""
63          # printf("TZ=UTC ./nilmtool.py %s\n", arg_string)
64          os.environ['NILMDB_URL'] = "http://localhost:32180/"
65          class stdio_wrapper:
66              def __init__(self, stdin, stdout, stderr):
67                  self.io = (stdin, stdout, stderr)
68              def __enter__(self):
69                  self.saved = ( sys.stdin, sys.stdout, sys.stderr )
70                  ( sys.stdin, sys.stdout, sys.stderr ) = self.io
71              def __exit__(self, type, value, traceback):
72                  ( sys.stdin, sys.stdout, sys.stderr ) = self.saved
73          # Empty input if none provided
74          if infile is None:
75              infile = UTF8StringIO("")
76          # Capture stderr
77          errfile = UTF8StringIO()
78          if outfile is None:
79              # If no output file, capture stdout with stderr
80              outfile = errfile
81          with stdio_wrapper(infile, outfile, errfile) as s:
82              try:
83                  # shlex doesn't support Unicode very well.  Encode the
84                  # string as UTF-8 explicitly before splitting.
85                  args = shlex.split(arg_string.encode('utf-8'))
86                  nilmdb.cmdline.Cmdline(args).run()
87                  sys.exit(0)
88              except SystemExit as e:
89                  exitcode = e.code
90          captured = nilmdb.utils.unicode.decode(outfile.getvalue())
91          self.captured = captured
92          self.exitcode = exitcode
93
94      def ok(self, arg_string, infile = None):
95          self.run(arg_string, infile)
96          if self.exitcode != 0:
97              self.dump()
98              eq_(self.exitcode, 0)
99
100     def fail(self, arg_string, infile = None,
101              exitcode = None, require_error = True):
102         self.run(arg_string, infile)
103         if exitcode is not None and self.exitcode != exitcode:
104             # Wrong exit code
105             self.dump()
106             eq_(self.exitcode, exitcode)
107         if self.exitcode == 0:
108             # Success, when we wanted failure
109             self.dump()
110             ne_(self.exitcode, 0)
111         # Make sure the output contains the word "error" at the
112         # beginning of a line, but only if an exitcode wasn't
113         # specified.
114         if require_error and not re.search("^error",
115                                            self.captured, re.MULTILINE):
116             raise AssertionError("command failed, but output doesn't "
117                                  "contain the string 'error'")
```

```
118
119     def contain(self, checkstring):
120         in_(checkstring, self.captured)
121
122     def match(self, checkstring):
123         eq_(checkstring, self.captured)
124
125     def matchfile(self, file):
126         # Captured data should match file contents exactly
127         with open(file) as f:
128             contents = f.read()
129             if contents != self.captured:
130                 print "--- reference file (first 1000 bytes):\n"
131                 print contents[0:1000] + "\n"
132                 print "--- captured data (first 1000 bytes):\n"
133                 print self.captured[0:1000] + "\n"
134                 zipped = itertools.izip_longest(contents, self.captured)
135                 for (n, (a, b)) in enumerate(zipped):
136                     if a != b:
137                         print "--- first difference is at offset", n
138                         print "--- reference:", repr(a)
139                         print "---  captured:", repr(b)
140                         break
141                 raise AssertionError("captured data doesn't match " + file)
142
143     def matchfilecount(self, file):
144         # Last line of captured data should match the number of
145         # non-commented lines in file
146         count = 0
147         with open(file) as f:
148             for line in f:
149                 if line[0] != '#':
150                     count += 1
151         eq_(self.captured.splitlines()[-1], sprintf("%d", count))
152
153     def dump(self):
154         printf("-----dump start-----\n%s-----dump end-----\n", self.captured)
155
156     def test_01_basic(self):
157
158         # help
159         self.ok("--help")
160         self.contain("usage:")
161
162         # help
163         self.ok("--version")
164         ver = self.captured
165         self.ok("list --version")
166         eq_(self.captured, ver)
167
168         # fail for no args
169         self.fail("")
170
171         # fail for no such option
172         self.fail("--nosuchoption")
173
174         # fail for bad command
175         self.fail("badcommand")
176
177         # try some URL constructions
178         self.fail("--url http://nosuchurl/ info")
179         self.contain("error connecting to server")
180
181         self.fail("--url nosuchurl info")
182         self.contain("error connecting to server")
183
184         self.fail("-u nosuchurl/foo info")
```

```
185         self.contain("error connecting to server")
186
187         self.fail("-u localhost:1 info")
188         self.contain("error connecting to server")
189
190         self.ok("-u localhost:32180 info")
191         self.ok("info")
192
193         # Duplicated arguments should fail, but this isn't implemented
194         # due to it being kind of a pain with argparse.
195         if 0:
196             self.fail("-u url1 -u url2 info")
197             self.contain("duplicated argument")
198
199             self.fail("list --detail --detail")
200             self.contain("duplicated argument")
201
202             self.fail("list --detail --path path1 --path path2")
203             self.contain("duplicated argument")
204
205             self.fail("extract --start 2000-01-01 --start 2001-01-02")
206             self.contain("duplicated argument")
207
208         # Verify that "help command" and "command --help" are identical
209         # for all commands.
210         self.fail("")
211         m = re.search(r"{(.*)}", self.captured)
212         for command in [""] + m.group(1).split(','):
213             self.ok(command + " --help")
214             cap1 = self.captured
215             self.ok("help " + command)
216             cap2 = self.captured
217             self.ok("help " + command + " asdf --url --zxcv -")
218             cap3 = self.captured
219             eq_(cap1, cap2)
220             eq_(cap2, cap3)
221
222     def test_02_parsetime(self):
223         os.environ['TZ'] = "America/New_York"
224         test = datetime_tz.datetime_tz.now()
225         u2ts = nilmdb.utils.time.unix_to_timestamp
226         parse_time = nilmdb.utils.time.parse_time
227         eq_(parse_time(str(test)), u2ts(test.totimestamp()))
228         test = u2ts(datetime_tz.datetime_tz.smartparse("20120405 1400-0400").
229                     totimestamp())
230         eq_(parse_time("hi there 20120405 1400-0400 testing! 123"), test)
231         eq_(parse_time("20120405 1800 UTC"), test)
232         eq_(parse_time("20120405 1400-0400 UTC"), test)
233         for badtime in [ "20120405 1400-9999", "hello", "-", "", "4:00" ]:
234             with assert_raises(ValueError):
235                 x = parse_time(badtime)
236         x = parse_time("now")
237         eq_(parse_time("snapshot-20120405-140000.raw.gz"), test)
238         eq_(parse_time("prep-20120405T1400"), test)
239         eq_(parse_time("1333648800.0"), test)
240         eq_(parse_time("1333648800000000"), test)
241         eq_(parse_time("@1333648800000000"), test)
242         eq_(parse_time("min"), nilmdb.utils.time.min_timestamp)
243         eq_(parse_time("max"), nilmdb.utils.time.max_timestamp)
244         with assert_raises(ValueError):
245             parse_time("@hashtag12345")
246
247     def test_03_info(self):
248         self.ok("info")
249         self.contain("Server URL: http://localhost:32180/")
250         self.contain("Client version: " + nilmdb.__version__)
251         self.contain("Server version: " + test_server.version)
```

```
252             self.contain("Server database path")
253             self.contain("Server disk space used by NilmDB")
254             self.contain("Server disk space used by other")
255             self.contain("Server disk space reserved")
256             self.contain("Server disk space free")
257
258         def test_04_createlist(self):
259             # Basic stream tests, like those in test_client.
260
261             # No streams
262             self.ok("list")
263             self.match("")
264
265             # Bad paths
266             self.fail("create foo/bar/baz float32_8")
267             self.contain("paths must start with /")
268
269             self.fail("create /foo float32_8")
270             self.contain("invalid path")
271             self.fail("create /newton/prep/ float32_8")
272             self.contain("invalid path")
273
274             self.fail("create /newton/_format/prep float32_8")
275             self.contain("path name is invalid")
276             self.fail("create /_format/newton/prep float32_8")
277             self.contain("path name is invalid")
278             self.fail("create /newton/prep/_format float32_8")
279             self.contain("path name is invalid")
280
281             # Bad layout type
282             self.fail("create /newton/prep NoSuchLayout")
283             self.contain("no such layout")
284             self.fail("create /newton/prep float32_0")
285             self.contain("no such layout")
286             self.fail("create /newton/prep float33_1")
287             self.contain("no such layout")
288
289             # Create a few streams
290             self.ok("create /newton/zzz/rawnotch uint16_9")
291             self.ok("create /newton/prep float32_8")
292             self.ok("create /newton/raw uint16_6")
293             self.ok("create /newton/raw~decim-1234 uint16_6")
294
295             # Create a stream that already exists
296             self.fail("create /newton/raw uint16_6")
297             self.contain("stream already exists at this path")
298
299             # Should not be able to create a stream with another stream as
300             # its parent
301             self.fail("create /newton/prep/blah float32_8")
302             self.contain("path is subdir of existing node")
303
304             # Should not be able to create a stream at a location that
305             # has other nodes as children
306             self.fail("create /newton/zzz float32_8")
307             self.contain("subdirs of this path already exist")
308
309             # Verify we got those 4 streams and they're returned in
310             # alphabetical order.
311             self.ok("list -l")
312             self.match("/newton/prep float32_8\n"
313                        "/newton/raw uint16_6\n"
314                        "/newton/raw~decim-1234 uint16_6\n"
315                        "/newton/zzz/rawnotch uint16_9\n")
316
317             # No decimated streams if -n specified
318             self.ok("list -n -l")
```

```
319             self.match("/newton/prep float32_8\n"
320                        "/newton/raw uint16_6\n"
321                        "/newton/zzz/rawnotch uint16_9\n")
322
323             # Delete that decimated stream
324             self.ok("destroy /newton/raw~decim-1234")
325
326             # Match just one type or one path.  Also check
327             # that --path is optional
328             self.ok("list --layout /newton/raw")
329             self.match("/newton/raw uint16_6\n")
330
331             # Wildcard matches
332             self.ok("list *zzz*")
333             self.match("/newton/zzz/rawnotch\n")
334
335             # reversed range
336             self.fail("list /newton/prep --start 2020-01-01 --end 2000-01-01")
337             self.contain("start must precede end")
338
339         def test_05_metadata(self):
340             # Set / get metadata
341             self.fail("metadata")
342             self.fail("metadata --get")
343
344             self.ok("metadata /newton/prep")
345             self.match("")
346
347             self.ok("metadata /newton/raw --get")
348             self.match("")
349
350             self.ok("metadata /newton/prep --set "
351                     "'description=The Data' "
352                     "v_scale=1.234")
353             self.ok("metadata /newton/raw --update "
354                     "'description=The Data'")
355             self.ok("metadata /newton/raw --update "
356                     "v_scale=1.234")
357
358             # various parsing tests
359             self.ok("metadata /newton/raw --update foo=")
360             self.fail("metadata /newton/raw --update =bar")
361             self.fail("metadata /newton/raw --update foo==bar")
362             self.fail("metadata /newton/raw --update foo;bar")
363
364             # errors
365             self.fail("metadata /newton/nosuchstream foo=bar")
366             self.contain("unrecognized arguments")
367             self.fail("metadata /newton/nosuchstream")
368             self.contain("No stream at path")
369             self.fail("metadata /newton/nosuchstream --set foo=bar")
370             self.contain("No stream at path")
371             self.fail("metadata /newton/nosuchstream --delete")
372             self.contain("No stream at path")
373
374             self.ok("metadata /newton/prep")
375             self.match("description=The Data\nv_scale=1.234\n")
376
377             self.ok("metadata /newton/prep --get")
378             self.match("description=The Data\nv_scale=1.234\n")
379
380             self.ok("metadata /newton/prep --get descr")
381             self.match("descr=\n")
382
383             self.ok("metadata /newton/prep --get description")
384             self.match("description=The Data\n")
385
```

```
386         self.ok("metadata /newton/prep --get description v_scale")
387         self.match("description=The Data\nv_scale=1.234\n")
388 
389         self.ok("metadata /newton/prep --set "
390                 "'description=The Data'")
391 
392         self.ok("metadata /newton/prep --get")
393         self.match("description=The Data\n")
394 
395         self.fail("metadata /newton/nosuchpath")
396         self.contain("No stream at path /newton/nosuchpath")
397 
398         self.ok("metadata /newton/prep --delete")
399         self.ok("metadata /newton/prep --get")
400         self.match("")
401         self.ok("metadata /newton/prep --set "
402                 "'description=The Data' "
403                 "v_scale=1.234")
404         self.ok("metadata /newton/prep --delete v_scale")
405         self.ok("metadata /newton/prep --get")
406         self.match("description=The Data\n")
407         self.ok("metadata /newton/prep --set description=")
408         self.ok("metadata /newton/prep --get")
409         self.match("")
410 
411     def test_06_insert(self):
412         self.ok("insert --help")
413 
414         self.fail("insert -s 2000 -e 2001 /foo/bar baz")
415         self.contain("error getting stream info")
416 
417         self.fail("insert -s 2000 -e 2001 /newton/prep baz")
418         self.match("error opening input file baz\n")
419 
420         self.fail("insert /newton/prep --timestamp -f -r 120")
421         self.contain("error extracting start time")
422 
423         self.fail("insert /newton/prep --timestamp -r 120")
424         self.contain("need --start or --filename")
425 
426         self.fail("insert /newton/prep "
427                   "tests/data/prep-20120323T1000")
428 
429         # insert pre-timestamped data, with bad times (non-monotonic)
430         os.environ['TZ'] = "UTC"
431         with open("tests/data/prep-20120323T1004-badtimes") as input:
432             self.fail("insert -s 20120323T1004 -e 20120323T1006 /newton/prep",
433                       input)
434             self.contain("error parsing input data")
435             self.contain("line 7")
436             self.contain("timestamp is not monotonically increasing")
437 
438         # insert pre-timestamped data, from stdin
439         os.environ['TZ'] = "UTC"
440         with open("tests/data/prep-20120323T1004-timestamped") as input:
441             self.ok("insert -s 20120323T1004 -e 20120323T1006 /newton/prep",
442                     input)
443 
444         # insert data with normal timestamper from filename
445         os.environ['TZ'] = "UTC"
446         self.ok("insert --timestamp -f --rate 120 /newton/prep "
447                 "tests/data/prep-20120323T1000")
448         self.fail("insert -t --filename /newton/prep "
449                 "tests/data/prep-20120323T1002")
450         self.contain("rate is needed")
451         self.ok("insert -t --filename --rate 120 /newton/prep "
452                 "tests/data/prep-20120323T1002")
```

```
453
454         # overlap
455         os.environ['TZ'] = "UTC"
456         self.fail("insert --timestamp -f --rate 120 /newton/prep "
457                   "tests/data/prep-20120323T1004")
458         self.contain("overlap")
459
460         # Just to help test more situations -- stop and restart
461         # the server now.  This tests nilmdb's interval caching,
462         # at the very least.
463         server_stop()
464         server_start()
465
466         # still an overlap if we specify a different start
467         os.environ['TZ'] = "America/New_York"
468         self.fail("insert -t -r 120 --start '03/23/2012 06:05:00' /newton/prep"
469                   " tests/data/prep-20120323T1004")
470         self.contain("overlap")
471
472         # wrong format
473         os.environ['TZ'] = "UTC"
474         self.fail("insert -t -r 120 -f /newton/raw "
475                   "tests/data/prep-20120323T1004")
476         self.contain("error parsing input data")
477         self.contain("can't parse value")
478
479         # too few rows per line
480         self.ok("create /insert/test float32_20")
481         self.fail("insert -t -r 120 -f /insert/test "
482                   "tests/data/prep-20120323T1004")
483         self.contain("error parsing input data")
484         self.contain("wrong number of values")
485         self.ok("destroy /insert/test")
486
487         # empty data does nothing
488         self.ok("insert -t -r 120 --start '03/23/2012 06:05:00' /newton/prep "
489                 "/dev/null")
490
491         # bad start time
492         self.fail("insert -t -r 120 --start 'whatever' /newton/prep /dev/null")
493
494         # Test negative times
495         self.ok("insert --start @-10000000000 --end @1000000001 /newton/prep"
496                 " tests/data/timestamped")
497         self.ok("extract -c /newton/prep --start min --end @1000000001")
498         self.match("8\n")
499         self.ok("remove /newton/prep --start min --end @1000000001")
500
501     def test_07_detail_extended(self):
502         # Just count the number of lines, it's probably fine
503         self.ok("list --detail")
504         lines_(self.captured, 8)
505
506         self.ok("list --detail *prep")
507         lines_(self.captured, 4)
508
509         self.ok("list --detail *prep --start='23 Mar 2012 10:02'")
510         lines_(self.captured, 3)
511
512         self.ok("list --detail *prep --start='23 Mar 2012 10:05'")
513         lines_(self.captured, 2)
514
515         self.ok("list --detail *prep --start='23 Mar 2012 10:05:15'")
516         lines_(self.captured, 2)
517         self.contain("10:05:15.000")
518
519         self.ok("list --detail *prep --start='23 Mar 2012 10:05:15.50'")
```

```
520         lines_(self.captured, 2)
521         self.contain("10:05:15.500")
522 
523         self.ok("list --detail *prep --start='23 Mar 2012 19:05:15.50'")
524         lines_(self.captured, 2)
525         self.contain("no intervals")
526 
527         self.ok("list --detail *prep --start='23 Mar 2012 10:05:15.50'"
528                 + " --end='23 Mar 2012 10:05:15.51'")
529         lines_(self.captured, 2)
530         self.contain("10:05:15.500")
531 
532         self.ok("list --detail")
533         lines_(self.captured, 8)
534 
535         # Verify the "raw timestamp" output
536         self.ok("list --detail *prep --timestamp-raw "
537                 "--start='23 Mar 2012 10:05:15.50'")
538         lines_(self.captured, 2)
539         self.contain("[ 1332497115500000 -> 1332497160000000 ]")
540 
541         # bad time
542         self.fail("list --detail *prep -T --start='9332497115.612'")
543         # good time
544         self.ok("list --detail *prep -T --start='1332497115.612'")
545         lines_(self.captured, 2)
546         self.contain("[ 1332497115612000 -> 1332497160000000 ]")
547 
548         # Check --ext output
549         self.ok("list --ext")
550         lines_(self.captured, 9)
551 
552         self.ok("list -E -T")
553         c = self.contain
554         c("\n  interval extents: 1332496800000000 -> 1332497160000000\n")
555         c("\n        total data: 43200 rows, 359.983336 seconds\n")
556         c("\n  interval extents: (no data)\n")
557         c("\n        total data: 0 rows, 0.000000 seconds\n")
558 
559         # Misc
560         self.fail("list --ext --start='23 Mar 2012 10:05:15.50'")
561         self.contain("--start and --end only make sense with --detail")
562 
563     def test_08_extract(self):
564         # nonexistent stream
565         self.fail("extract /no/such/foo --start 2000-01-01 --end 2020-01-01")
566         self.contain("error getting stream info")
567 
568         # reversed range
569         self.fail("extract -a /newton/prep --start 2020-01-01 --end 2000-01-01")
570         self.contain("start is after end")
571 
572         # empty ranges return error 2
573         self.fail("extract -a /newton/prep " +
574                   "--start '23 Mar 2012 20:00:30' " +
575                   "--end '23 Mar 2012 20:00:31'",
576                   exitcode = 2, require_error = False)
577         self.contain("no data")
578         self.fail("extract -a /newton/prep " +
579                   "--start '23 Mar 2012 20:00:30.000001' " +
580                   "--end '23 Mar 2012 20:00:30.000002'",
581                   exitcode = 2, require_error = False)
582         self.contain("no data")
583         self.fail("extract -a /newton/prep " +
584                   "--start '23 Mar 2022 10:00:30' " +
585                   "--end '23 Mar 2022 10:00:31'",
586                   exitcode = 2, require_error = False)
```

```
587             self.contain("no data")
588
589             # but are ok if we're just counting results
590             self.ok("extract --count /newton/prep " +
591                     "--start '23 Mar 2012 20:00:30' " +
592                     "--end '23 Mar 2012 20:00:31'")
593             self.match("0\n")
594             self.ok("extract -c /newton/prep " +
595                     "--start '23 Mar 2012 20:00:30.000001' " +
596                     "--end '23 Mar 2012 20:00:30.000002'")
597             self.match("0\n")
598
599             # Check various dumps against stored copies of how they should appear
600             def test(file, start, end, extra=""):
601                 self.ok("extract " + extra + " /newton/prep " +
602                         "--start '23 Mar 2012 " + start + "' " +
603                         "--end '23 Mar 2012 " + end + "'")
604                 self.matchfile("tests/data/extract-" + str(file))
605                 self.ok("extract --count " + extra + " /newton/prep " +
606                         "--start '23 Mar 2012 " + start + "' " +
607                         "--end '23 Mar 2012 " + end + "'")
608                 self.matchfilecount("tests/data/extract-" + str(file))
609             test(1, "10:00:30", "10:00:31", extra="-a")
610             test(1, "10:00:30.000000", "10:00:31", extra="-a")
611             test(2, "10:00:30.000001", "10:00:31")
612             test(2, "10:00:30.008333", "10:00:31")
613             test(3, "10:00:30.008333", "10:00:30.008334")
614             test(3, "10:00:30.008333", "10:00:30.016667")
615             test(4, "10:00:30.008333", "10:00:30.025")
616             test(5, "10:00:30", "10:00:31", extra="--annotate --bare")
617             test(6, "10:00:30", "10:00:31", extra="-b")
618             test(7, "10:00:30", "10:00:30.999", extra="-a -T")
619             test(7, "10:00:30", "10:00:30.999", extra="-a --timestamp-raw")
620             test(8, "10:01:59.9", "10:02:00.1", extra="--markup")
621             test(8, "10:01:59.9", "10:02:00.1", extra="-m")
622
623             # all data put in by tests
624             self.ok("extract -a /newton/prep --start min --end max")
625             lines_(self.captured, 43204)
626             self.ok("extract -c /newton/prep --start 2000-01-01 --end 2020-01-01")
627             self.match("43200\n")
628
629             # test binary mode
630             self.fail("extract -c -B /newton/prep -s min -e max")
631             self.contain("binary cannot be combined")
632             self.fail("extract -m -B /newton/prep -s min -e max")
633             self.contain("binary cannot be combined")
634             self.ok("extract -B /newton/prep -s min -e max")
635             eq_(len(self.captured), 43200 * (8 + 8*4))
636
637             # markup for 3 intervals, plus extra markup lines whenever we had
638             # a "restart" from the nilmdb.stream_extract function
639             self.ok("extract -m /newton/prep --start 2000-01-01 --end 2020-01-01")
640             lines_(self.captured, 43210)
641
642         def test_09_truncated(self):
643             # Test truncated responses by overriding the nilmdb max_results
644             server_stop()
645             server_start(max_results = 2)
646             self.ok("list --detail")
647             lines_(self.captured, 8)
648             server_stop()
649             server_start()
650
651         def test_10_remove(self):
652             # Removing data
653
```

```
654         # Try nonexistent stream
655         self.fail("remove /no/such/foo --start 2000-01-01 --end 2020-01-01")
656         self.contain("no stream matched path")
657 
658         # empty or backward ranges return errors
659         self.fail("remove /newton/prep --start 2020-01-01 --end 2000-01-01")
660         self.contain("start must precede end")
661 
662         self.fail("remove /newton/prep " +
663                   "--start '23 Mar 2012 10:00:30' " +
664                   "--end '23 Mar 2012 10:00:30'")
665         self.contain("start must precede end")
666         self.fail("remove /newton/prep " +
667                   "--start '23 Mar 2012 10:00:30.000001' " +
668                   "--end '23 Mar 2012 10:00:30.000001'")
669         self.contain("start must precede end")
670         self.fail("remove /newton/prep " +
671                   "--start '23 Mar 2022 10:00:30' " +
672                   "--end '23 Mar 2022 10:00:30'")
673         self.contain("start must precede end")
674 
675         # Verbose
676         self.ok("remove -c /newton/prep " +
677                 "--start '23 Mar 2022 20:00:30' " +
678                 "--end '23 Mar 2022 20:00:31'")
679         self.match("0\n")
680         self.ok("remove --count /newton/prep " +
681                 "--start '23 Mar 2022 20:00:30' " +
682                 "--end '23 Mar 2022 20:00:31'")
683         self.match("0\n")
684         self.ok("remove -c /newton/prep /newton/pre* " +
685                 "--start '23 Mar 2022 20:00:30' " +
686                 "--end '23 Mar 2022 20:00:31'")
687         self.match("Removing from /newton/prep\n0\n" +
688                    "Removing from /newton/prep\n0\n")
689 
690         # Make sure we have the data we expect
691         self.ok("list -l --detail /newton/prep")
692         self.match("/newton/prep float32_8\n" +
693                    "  [ Fri, 23 Mar 2012 10:00:00.000000 +0000"
694                    " -> Fri, 23 Mar 2012 10:01:59.991668 +0000 ]\n"
695                    "  [ Fri, 23 Mar 2012 10:02:00.000000 +0000"
696                    " -> Fri, 23 Mar 2012 10:03:59.991668 +0000 ]\n"
697                    "  [ Fri, 23 Mar 2012 10:04:00.000000 +0000"
698                    " -> Fri, 23 Mar 2012 10:06:00.000000 +0000 ]\n")
699 
700         # Remove various chunks of prep data and make sure
701         # they're gone.
702         self.ok("remove -c /newton/prep " +
703                 "--start '23 Mar 2012 10:00:30' " +
704                 "--end '23 Mar 2012 10:00:40'")
705         self.match("1200\n")
706 
707         self.ok("remove -c /newton/prep " +
708                 "--start '23 Mar 2012 10:00:10' " +
709                 "--end '23 Mar 2012 10:00:20'")
710         self.match("1200\n")
711 
712         self.ok("remove -c /newton/prep " +
713                 "--start '23 Mar 2012 10:00:05' " +
714                 "--end '23 Mar 2012 10:00:25'")
715         self.match("1200\n")
716 
717         self.ok("remove -c /newton/prep " +
718                 "--start '23 Mar 2012 10:03:50' " +
719                 "--end '23 Mar 2012 10:06:50'")
720         self.match("15600\n")
```

```
721
722         self.ok("extract -c /newton/prep --start 2000-01-01 --end 2020-01-01")
723         self.match("24000\n")
724
725         # See the missing chunks in list output
726         self.ok("list --layout --detail /newton/prep")
727         self.match("/newton/prep float32_8\n" +
728                    "  [ Fri, 23 Mar 2012 10:00:00.000000 +0000"
729                    " -> Fri, 23 Mar 2012 10:00:05.000000 +0000 ]\n"
730                    "  [ Fri, 23 Mar 2012 10:00:25.000000 +0000"
731                    " -> Fri, 23 Mar 2012 10:00:30.000000 +0000 ]\n"
732                    "  [ Fri, 23 Mar 2012 10:00:40.000000 +0000"
733                    " -> Fri, 23 Mar 2012 10:01:59.991668 +0000 ]\n"
734                    "  [ Fri, 23 Mar 2012 10:02:00.000000 +0000"
735                    " -> Fri, 23 Mar 2012 10:03:50.000000 +0000 ]\n")
736
737         # Remove all data, verify it's missing
738         self.ok("remove /newton/prep --start 2000-01-01 --end 2020-01-01")
739         self.match("")  # no count requested this time
740         self.ok("list -l --detail /newton/prep")
741         self.match("/newton/prep float32_8\n" +
742                    "  (no intervals)\n")
743
744         # Reinsert some data, to verify that no overlaps with deleted
745         # data are reported
746         for minute in ["0", "2"]:
747             self.ok("insert --timestamp -f --rate 120 /newton/prep"
748                     " tests/data/prep-20120323T100" + minute)
749
750     def test_11_destroy(self):
751         # Delete records
752         self.ok("destroy --help")
753
754         self.fail("destroy")
755         self.contain("too few arguments")
756
757         self.fail("destroy /no/such/stream")
758         self.contain("no stream matched path")
759
760         self.fail("destroy -R /no/such/stream")
761         self.contain("no stream matched path")
762
763         self.fail("destroy asdfasdf")
764         self.contain("no stream matched path")
765
766         # From previous tests, we have:
767         self.ok("list -l")
768         self.match("/newton/prep float32_8\n"
769                    "/newton/raw uint16_6\n"
770                    "/newton/zzz/rawnotch uint16_9\n")
771
772         # Notice how they're not empty
773         self.ok("list --detail")
774         lines_(self.captured, 7)
775
776         # Fail to destroy because intervals still present
777         self.fail("destroy /newton/prep")
778         self.contain("all intervals must be removed")
779         self.ok("list --detail")
780         lines_(self.captured, 7)
781
782         # Destroy for real
783         self.ok("destroy -R /n*/prep")
784         self.ok("list -l")
785         self.match("/newton/raw uint16_6\n"
786                    "/newton/zzz/rawnotch uint16_9\n")
787
```

```
788         self.ok("destroy /newton/zzz/rawnotch")
789         self.ok("list -l")
790         self.match("/newton/raw uint16_6\n")
791 
792         self.ok("destroy /newton/raw")
793         self.ok("create /newton/raw uint16_6")
794         # Specify --remove with no data
795         self.ok("destroy --remove /newton/raw")
796         self.ok("list")
797         self.match("")
798 
799         # Re-create a previously deleted location, and some new ones
800         rebuild = [ "/newton/prep", "/newton/zzz",
801                     "/newton/raw", "/newton/asdf/qwer" ]
802         for path in rebuild:
803             # Create the path
804             self.ok("create " + path + " float32_8")
805             self.ok("list")
806             self.contain(path)
807             # Make sure it was created empty
808             self.ok("list --detail " + path)
809             self.contain("(no intervals)")
810 
811     def test_12_unicode(self):
812         # Unicode paths.
813         self.ok("destroy /newton/asdf/qwer")
814         self.ok("destroy /newton/prep /newton/raw")
815         self.ok("destroy /newton/zzz")
816 
817         self.ok(u"create /düsseldorf/raw uint16_6")
818         self.ok("list -l --detail")
819         self.contain(u"/düsseldorf/raw uint16_6")
820         self.contain("(no intervals)")
821 
822         # Unicode metadata
823         self.ok(u"metadata /düsseldorf/raw --set α=beta γδ'='")
824         self.ok(u"metadata /düsseldorf/raw --update αβ'= ε τ α'")
825         self.ok(u"metadata /düsseldorf/raw")
826         self.match(u"αβ= ε τ α\γnδ=\n")
827 
828         self.ok(u"destroy /düsseldorf/raw")
829 
830     def test_13_files(self):
831         # Test BulkData's ability to split into multiple files,
832         # by forcing the file size to be really small.
833         server_stop()
834         server_start(bulkdata_args = { "file_size" : 920, # 23 rows per file
835                                        "files_per_dir" : 3 })
836 
837         # Fill data
838         self.ok("create /newton/prep float32_8")
839         os.environ['TZ'] = "UTC"
840         with open("tests/data/prep-20120323T1004-timestamped") as input:
841             self.ok("insert -s 20120323T1004 -e 20120323T1006 /newton/prep",
842                     input)
843 
844         # Extract it
845         self.ok("extract /newton/prep --start '2000-01-01' " +
846                 "--end '2012-03-23 10:04:01'")
847         lines_(self.captured, 120)
848         self.ok("extract /newton/prep --start '2000-01-01' " +
849                 "--end '2022-03-23 10:04:01'")
850         lines_(self.captured, 14400)
851 
852         # Make sure there were lots of files generated in the database
853         # dir
854         nfiles = 0
```

```
855         for (dirpath, dirnames, filenames) in os.walk(testdb):
856             nfiles += len(filenames)
857         assert(nfiles > 500)
858 
859         # Make sure we can restart the server with a different file
860         # size and have it still work
861         server_stop()
862         server_start()
863         self.ok("extract /newton/prep --start '2000-01-01' " +
864                 "--end '2022-03-23 10:04:01'")
865         lines_(self.captured, 14400)
866 
867         # Now recreate the data one more time and make sure there are
868         # fewer files.
869         self.ok("destroy --remove /newton/prep")
870         self.fail("destroy /newton/prep") # already destroyed
871         self.ok("create /newton/prep float32_8")
872         os.environ['TZ'] = "UTC"
873         with open("tests/data/prep-20120323T1004-timestamped") as input:
874             self.ok("insert -s 20120323T1004 -e 20120323T1006 /newton/prep",
875                     input)
876         nfiles = 0
877         for (dirpath, dirnames, filenames) in os.walk(testdb):
878             nfiles += len(filenames)
879         lt_(nfiles, 50)
880         self.ok("destroy -R /newton/prep") # destroy again
881 
882     def test_14_remove_files(self):
883         # Test BulkData's ability to remove when data is split into
884         # multiple files.  Should be a fairly comprehensive test of
885         # remove functionality.
886         # Also limit max_removals, to cover more functionality.
887         server_stop()
888         server_start(max_removals = 4321,
889                      bulkdata_args = { "file_size" : 920, # 23 rows per file
890                                        "files_per_dir" : 3 })
891 
892         # Insert data.  Just for fun, insert out of order
893         self.ok("create /newton/prep float32_8")
894         os.environ['TZ'] = "UTC"
895         self.ok("insert -t --filename --rate 120 /newton/prep "
896                 "tests/data/prep-20120323T1002")
897         self.ok("insert -t --filename --rate 120 /newton/prep "
898                 "tests/data/prep-20120323T1000")
899 
900         # Should take up about 2.8 MB here (including directory entries)
901         du_before = nilmdb.utils.diskusage.du(testdb)
902 
903         # Make sure we have the data we expect
904         self.ok("list -l --detail")
905         self.match("/newton/prep float32_8\n" +
906                    "  [ Fri, 23 Mar 2012 10:00:00.000000 +0000"
907                    " -> Fri, 23 Mar 2012 10:01:59.991668 +0000 ]\n"
908                    "  [ Fri, 23 Mar 2012 10:02:00.000000 +0000"
909                    " -> Fri, 23 Mar 2012 10:03:59.991668 +0000 ]\n")
910 
911         # Remove various chunks of prep data and make sure
912         # they're gone.
913         self.ok("extract -c /newton/prep --start 2000-01-01 --end 2020-01-01")
914         self.match("28800\n")
915 
916         self.ok("remove -c /newton/prep " +
917                 "--start '23 Mar 2012 10:00:30' " +
918                 "--end '23 Mar 2012 10:03:30'")
919         self.match("21600\n")
920 
921         self.ok("remove -c /newton/prep " +
```

```
922                 "--start '23 Mar 2012 10:00:10' " +
923                 "--end '23 Mar 2012 10:00:20'")
924         self.match("1200\n")
925
926         self.ok("remove -c /newton/prep " +
927                 "--start '23 Mar 2012 10:00:05' " +
928                 "--end '23 Mar 2012 10:00:25'")
929         self.match("1200\n")
930
931         self.ok("remove -c /newton/prep " +
932                 "--start '23 Mar 2012 10:03:50' " +
933                 "--end '23 Mar 2012 10:06:50'")
934         self.match("1200\n")
935
936         self.ok("extract -c /newton/prep --start 2000-01-01 --end 2020-01-01")
937         self.match("3600\n")
938
939         # See the missing chunks in list output
940         self.ok("list -l --detail")
941         self.match("/newton/prep float32_8\n" +
942                    "  [ Fri, 23 Mar 2012 10:00:00.000000 +0000"
943                    " -> Fri, 23 Mar 2012 10:00:05.000000 +0000 ]\n"
944                    "  [ Fri, 23 Mar 2012 10:00:25.000000 +0000"
945                    " -> Fri, 23 Mar 2012 10:00:30.000000 +0000 ]\n"
946                    "  [ Fri, 23 Mar 2012 10:03:30.000000 +0000"
947                    " -> Fri, 23 Mar 2012 10:03:50.000000 +0000 ]\n")
948
949         # We have 1/8 of the data that we had before, so the file size
950         # should have dropped below 1/4 of what it used to be
951         du_after = nilmdb.utils.diskusage.du(testdb)
952         lt_(du_after, (du_before / 4))
953
954         # Remove anything that came from the 10:02 data file
955         self.ok("remove /newton/prep " +
956                 "--start '23 Mar 2012 10:02:00' --end '2020-01-01'")
957
958         # Re-insert 19 lines from that file, then remove them again.
959         # With the specific file_size above, this will cause the last
960         # file in the bulk data storage to be exactly file_size large,
961         # so removing the data should also remove that last file.
962         self.ok("insert --timestamp -f --rate 120 /newton/prep " +
963                 "tests/data/prep-20120323T1002-first19lines")
964         self.ok("remove /newton/prep " +
965                 "--start '23 Mar 2012 10:02:00' --end '2020-01-01'")
966
967         # Shut down and restart server, to force nrows to get refreshed.
968         server_stop()
969         server_start()
970
971         # Re-add the full 10:02 data file.  This tests adding new data once
972         # we removed data near the end.
973         self.ok("insert -t -f -r 120 /newton/prep "
974                 "tests/data/prep-20120323T1002")
975
976         # See if we can extract it all
977         self.ok("extract /newton/prep --start 2000-01-01 --end 2020-01-01")
978         lines_(self.captured, 15600)
979
980     def test_15_intervals_diff(self):
981         # Test "intervals" and "intervals --diff" command.
982         os.environ['TZ'] = "UTC"
983
984         self.ok("create /diff/1 uint8_1")
985         self.match("")
986         self.ok("intervals /diff/1")
987         self.match("")
988         self.ok("intervals /diff/1 --diff /diff/1")
```

```
989             self.match("")
990             self.ok("intervals --diff /diff/1 /diff/1")
991             self.match("")
992             self.fail("intervals /diff/2")
993             self.fail("intervals /diff/1 -d /diff/2")
994
995             self.ok("create /diff/2 uint8_1")
996             self.ok("intervals -T /diff/1 -d /diff/2")
997             self.match("")
998             self.ok("insert -s 01-01-2000 -e 01-01-2001 /diff/1 /dev/null")
999
1000            self.ok("intervals /diff/1")
1001            self.match("[ Sat, 01 Jan 2000 00:00:00.000000 +0000 -"
1002                       "> Mon, 01 Jan 2001 00:00:00.000000 +0000 ]\n")
1003
1004            self.ok("intervals /diff/1 -d /diff/2")
1005            self.match("[ Sat, 01 Jan 2000 00:00:00.000000 +0000 -"
1006                       "> Mon, 01 Jan 2001 00:00:00.000000 +0000 ]\n")
1007
1008            self.ok("insert -s 01-01-2000 -e 01-01-2001 /diff/2 /dev/null")
1009            self.ok("intervals /diff/1 -d /diff/2")
1010            self.match("")
1011
1012            self.ok("insert -s 01-01-2001 -e 01-01-2002 /diff/1 /dev/null")
1013            self.ok("insert -s 01-01-2002 -e 01-01-2003 /diff/2 /dev/null")
1014            self.ok("intervals /diff/1 -d /diff/2")
1015            self.match("[ Mon, 01 Jan 2001 00:00:00.000000 +0000 -"
1016                       "> Tue, 01 Jan 2002 00:00:00.000000 +0000 ]\n")
1017
1018            self.ok("insert -s 01-01-2004 -e 01-01-2005 /diff/1 /dev/null")
1019            self.ok("intervals /diff/1 -d /diff/2")
1020            self.match("[ Mon, 01 Jan 2001 00:00:00.000000 +0000 -"
1021                       "> Tue, 01 Jan 2002 00:00:00.000000 +0000 ]\n"
1022                       "[ Thu, 01 Jan 2004 00:00:00.000000 +0000 -"
1023                       "> Sat, 01 Jan 2005 00:00:00.000000 +0000 ]\n")
1024
1025            self.fail("intervals -s 01-01-2003 -e 01-01-2000 /diff/1 -d /diff/2")
1026            self.ok("intervals -s 01-01-2003 -e 01-01-2008 /diff/1 -d /diff/2")
1027            self.match("[ Thu, 01 Jan 2004 00:00:00.000000 +0000 -"
1028                       "> Sat, 01 Jan 2005 00:00:00.000000 +0000 ]\n")
1029
1030            # optimize
1031            self.ok("insert -s 01-01-2002 -e 01-01-2004 /diff/1 /dev/null")
1032            self.ok("intervals /diff/1")
1033            self.match("[ Sat, 01 Jan 2000 00:00:00.000000 +0000 -"
1034                       "> Thu, 01 Jan 2004 00:00:00.000000 +0000 ]\n"
1035                       "[ Thu, 01 Jan 2004 00:00:00.000000 +0000 -"
1036                       "> Sat, 01 Jan 2005 00:00:00.000000 +0000 ]\n")
1037            self.ok("intervals /diff/1 --optimize")
1038            self.ok("intervals /diff/1 -o")
1039            self.match("[ Sat, 01 Jan 2000 00:00:00.000000 +0000 -"
1040                       "> Sat, 01 Jan 2005 00:00:00.000000 +0000 ]\n")
1041
1042            self.ok("destroy -R /diff/1")
1043            self.ok("destroy -R /diff/2")
1044
1045        def test_16_rename(self):
1046            # Test renaming.  Force file size smaller so we get more files
1047            server_stop()
1048            recursive_unlink(testdb)
1049            server_start(bulkdata_args = { "file_size" : 920, # 23 rows per file
1050                                           "files_per_dir" : 3 })
1051
1052
1053            # Fill data
1054            self.ok("create /newton/prep float32_8")
1055            os.environ['TZ'] = "UTC"
```

```
        with open("tests/data/prep-20120323T1004-timestamped") as input:
            self.ok("insert -s 20120323T1004 -e 20120323T1006 /newton/prep",
                    input)

        # Extract it
        self.ok("extract /newton/prep --start '2000-01-01' " +
                "--end '2012-03-23 10:04:01'")
        extract_before = self.captured

        def check_path(*components):
            # Verify the paths look right on disk
            seek = os.path.join(testdb, "data", *components)
            for (dirpath, dirnames, filenames) in os.walk(testdb):
                if "_format" in filenames:
                    if dirpath == seek:
                        break
                    raise AssertionError("data also found at " + dirpath)
            else:
                raise AssertionError("data not found at " + seek)
            # Verify "list" output
            self.ok("list -l")
            self.match("/" + "/".join(components) + " float32_8\n")

        # Lots of renames
        check_path("newton", "prep")

        self.fail("rename /newton/prep /newton/prep")
        self.contain("old and new paths are the same")
        check_path("newton", "prep")
        self.fail("rename /newton/prep /newton")
        self.contain("path must contain at least one folder")
        self.fail("rename /newton/prep /newton/prep/")
        self.contain("invalid path")
        self.ok("rename /newton/prep /newton/foo/1")
        check_path("newton", "foo", "1")
        self.ok("rename /newton/foo/1 /newton/foo")
        check_path("newton", "foo")
        self.ok("rename /newton/foo /totally/different/thing")
        check_path("totally", "different", "thing")
        self.ok("rename /totally/different/thing /totally/something")
        check_path("totally", "something")
        self.ok("rename /totally/something /totally/something/cool")
        check_path("totally", "something", "cool")
        self.ok("rename /totally/something/cool /foo/bar")
        check_path("foo", "bar")
        self.ok("create /xxx/yyy/zzz float32_8")
        self.fail("rename /foo/bar /xxx/yyy")
        self.contain("subdirs of this path already exist")
        self.fail("rename /foo/bar /xxx/yyy/zzz")
        self.contain("stream already exists at this path")
        self.fail("rename /foo/bar /xxx/yyy/zzz/www")
        self.contain("path is subdir of existing node")
        self.ok("rename /foo/bar /xxx/yyy/mmm")
        self.ok("destroy -R /xxx/yyy/zzz")
        check_path("xxx", "yyy", "mmm")

        # Extract it at the final path
        self.ok("extract /xxx/yyy/mmm --start '2000-01-01' " +
                "--end '2012-03-23 10:04:01'")
        eq_(self.captured, extract_before)

        self.ok("destroy -R /xxx/yyy/mmm")

        # Make sure temporary rename dirs weren't left around
        for (dirpath, dirnames, filenames) in os.walk(testdb):
            if "rename-" in dirpath:
                raise AssertionError("temporary directories not cleaned up")
```

```
            if "totally" in dirpath or "newton" in dirpath:
                raise AssertionError("old directories not cleaned up")

        server_stop()
        server_start()
```

Appendix D

NilmRun Implementation

This appendix contains the implementation of NilmRun, an HTTP interface that faciliates execution of Python code and other software on a remote machine, including log output capture and process lifecycle management. This code pulls in a number of support modules from NilmDB, which must be installed. A test suite is included to verify correctness.

D.1 Server

Listing D-1: `nilmrun/server.py`: HTTP server interface and WSGI application server. This dispatches HTTP requests, and provides setup routines for both the standalone CherryPy server and the Apache WSGI application.

Git repository: `https://git.jim.sh/jim/lees/nilmrun.git`
Filename: `nilmrun/server.py`
Revision: `38c3e67cf9f1a9912f94c0f8cad44841d9c48396`

```
1  """CherryPy-based server for running NILM filters via HTTP"""
2
3  import cherrypy
4  import sys
5  import os
6  import socket
7  import simplejson as json
8  import traceback
9  import time
10
11 import nilmdb
12 from nilmdb.utils.printf import *
13 from nilmdb.server.serverutil import (
```

```
        chunked_response,
        response_type,
        workaround_cp_bug_1200,
        exception_to_httperror,
        CORS_allow,
        json_to_request_params,
        json_error_page,
        cherrypy_start,
        cherrypy_stop,
        bool_param,
        )
from nilmdb.utils import serializer_proxy
import nilmrun
import nilmrun.testfilter

# Add CORS_allow tool
cherrypy.tools.CORS_allow = cherrypy.Tool('on_start_resource', CORS_allow)

# CherryPy apps
class App(object):
    """Root application for NILM runner"""

    def __init__(self):
        pass

    # /
    @cherrypy.expose
    def index(self):
        cherrypy.response.headers['Content-Type'] = 'text/plain'
        msg = sprintf("This is NilmRun version %s, running on host %s.\n",
                      nilmrun.__version__, socket.getfqdn())
        return msg

    # /favicon.ico
    @cherrypy.expose
    def favicon_ico(self):
        raise cherrypy.NotFound()

    # /version
    @cherrypy.expose
    @cherrypy.tools.json_out()
    def version(self):
        return nilmrun.__version__

class AppProcess(object):

    def __init__(self, manager):
        self.manager = manager

    def process_status(self, pid):
        # We need to convert the log (which is bytes) to Unicode
        # characters, in order to send it via JSON.  Treat it as UTF-8
        # but replace invalid characters with markers.
        log = self.manager[pid].log.decode('utf-8', errors='replace')
        return {
            "pid": pid,
            "alive": self.manager[pid].alive,
            "exitcode": self.manager[pid].exitcode,
            "start_time": self.manager[pid].start_time,
            "log": log
            }

    # /process/status
    @cherrypy.expose
    @cherrypy.tools.json_out()
    def status(self, pid, clear = False):
        """Return status about a process.  If clear = True, also clear
```

```
81          the log."""
82          clear = bool_param(clear)
83          if pid not in self.manager:
84              raise cherrypy.HTTPError("404 Not Found", "No such PID")
85          status = self.process_status(pid)
86          if clear:
87              self.manager[pid].clear_log()
88          return status
89
90      # /process/list
91      @cherrypy.expose
92      @cherrypy.tools.json_out()
93      def list(self):
94          """Return a list of processes in the manager."""
95          return list(self.manager)
96
97      # /process/info
98      @cherrypy.expose
99      @cherrypy.tools.json_out()
100     def info(self):
101         """Return detailed CPU and memory info about the system and
102         all processes"""
103         return self.manager.get_info()
104
105     # /process/remove
106     @cherrypy.expose
107     @cherrypy.tools.json_in()
108     @cherrypy.tools.json_out()
109     @cherrypy.tools.CORS_allow(methods = ["POST"])
110     def remove(self, pid):
111         """Remove a process from the manager, killing it if necessary."""
112         if pid not in self.manager:
113             raise cherrypy.HTTPError("404 Not Found", "No such PID")
114         if not self.manager.terminate(pid): # pragma: no cover
115             raise cherrypy.HTTPError("503 Service Unavailable",
116                                      "Failed to stop process")
117         status = self.process_status(pid)
118         self.manager.remove(pid)
119         return status
120
121 class AppRun(object):
122     def __init__(self, manager):
123         self.manager = manager
124
125     # /run/command
126     @cherrypy.expose
127     @cherrypy.tools.json_in()
128     @cherrypy.tools.json_out()
129     @exception_to_httperror(nilmrun.processmanager.ProcessError)
130     @cherrypy.tools.CORS_allow(methods = ["POST"])
131     def command(self, argv):
132         """Execute an arbitrary program on the server.  argv is a
133         list of the program and its arguments: 'argv[0]' is the program
134         and 'argv[1:]' are arguments"""
135         if not isinstance(argv, list):
136             raise cherrypy.HTTPError("400 Bad Request",
137                                      "argv must be a list of strings")
138         return self.manager.run_command(argv)
139
140     # /run/code
141     @cherrypy.expose
142     @cherrypy.tools.json_in()
143     @cherrypy.tools.json_out()
144     @exception_to_httperror(nilmrun.processmanager.ProcessError)
145     @cherrypy.tools.CORS_allow(methods = ["POST"])
146     def code(self, code, args = None):
147         """Execute arbitrary Python code.  'code' is a formatted string.
```

```
        It will be run as if it were written into a Python file and
        executed.  'args' is a list of strings, and they are passed
        on the command line as additional arguments (i.e., they end up
        in sys.argv[1:])"""
        if args is None:
            args = []
        if not isinstance(args, list):
            raise cherrypy.HTTPError("400 Bad Request",
                                     "args must be a list of strings")
        return self.manager.run_code(code, args)

class Server(object):
    def __init__(self, host = '127.0.0.1', port = 8080,
                 embedded = True,         # hide diagnostics and output, etc
                 force_traceback = False, # include traceback in all errors
                 basepath = '',           # base URL path for cherrypy.tree
                 ):
        self.embedded = embedded

        # Build up global server configuration
        cherrypy.config.update({
            'server.socket_host': host,
            'server.socket_port': port,
            'engine.autoreload_on': False,
            'server.max_request_body_size': 8*1024*1024,
            })
        if self.embedded:
            cherrypy.config.update({ 'environment': 'embedded' })

        # Build up application specific configuration
        app_config = {}
        app_config.update({
            'error_page.default': self.json_error_page,
            })

        # Some default headers to just help identify that things are working
        app_config.update({ 'response.headers.X-Jim-Is-Awesome': 'yeah' })

        # Set up Cross-Origin Resource Sharing (CORS) handler so we
        # can correctly respond to browsers' CORS preflight requests.
        # This also limits verbs to GET and HEAD by default.
        app_config.update({ 'tools.CORS_allow.on': True,
                            'tools.CORS_allow.methods': ['GET', 'HEAD'] })

        # Configure the 'json_in' tool to also allow other content-types
        # (like x-www-form-urlencoded), and to treat JSON as a dict that
        # fills requests.param.
        app_config.update({ 'tools.json_in.force': False,
                            'tools.json_in.processor': json_to_request_params })

        # Send tracebacks in error responses.  They're hidden by the
        # error_page function for client errors (code 400-499).
        app_config.update({ 'request.show_tracebacks' : True })
        self.force_traceback = force_traceback

        # Patch CherryPy error handler to never pad out error messages.
        # This isn't necessary, but then again, neither is padding the
        # error messages.
        cherrypy._cperror._ie_friendly_error_sizes = {}

        # The manager maintains internal state and isn't necessarily
        # thread-safe, so wrap it in the serializer.
        manager = serializer_proxy(nilmrun.processmanager.ProcessManager)()

        # Build up the application and mount it
        self._manager = manager
        root = App()
```

```
215         root.process = AppProcess(manager)
216         root.run = AppRun(manager)
217         cherrypy.tree.apps = {}
218         cherrypy.tree.mount(root, basepath, config = { "/" : app_config })
219 
220         # Set up the WSGI application pointer for external programs
221         self.wsgi_application = cherrypy.tree
222 
223     def json_error_page(self, status, message, traceback, version):
224         """Return a custom error page in JSON so the client can parse it"""
225         return json_error_page(status, message, traceback, version,
226                                self.force_traceback)
227 
228     def start(self, blocking = False, event = None):
229         cherrypy_start(blocking, event, self.embedded)
230 
231     def stop(self):
232         cherrypy_stop()
233 
234 # Multiple processes and threads should be OK here, but we'll still
235 # follow the NilmDB approach of having just one globally initialized
236 # copy of the server object.
237 _wsgi_server = None
238 def wsgi_application(basepath): # pragma: no cover
239     """Return a WSGI application object.
240 
241     'basepath' is the URL path of the application base, which
242     is the same as the first argument to Apache's WSGIScriptAlias
243     directive.
244     """
245     def application(environ, start_response):
246         global _wsgi_server
247         if _wsgi_server is None:
248             # Try to start the server
249             try:
250                 _wsgi_server = nilmrun.server.Server(
251                     embedded = True,
252                     basepath = basepath.rstrip('/'))
253             except Exception:
254                 # Build an error message on failure
255                 import pprint
256                 err = sprintf("Initializing nilmrun failed:\n\n",
257                               dbpath)
258                 err += traceback.format_exc()
259                 try:
260                     import pwd
261                     import grp
262                     err += sprintf("\nRunning as: uid=%d (%s), gid=%d (%s) "
263                                    "on host %s, pid %d\n",
264                                    os.getuid(), pwd.getpwuid(os.getuid())[0],
265                                    os.getgid(), grp.getgrgid(os.getgid())[0],
266                                    socket.gethostname(), os.getpid())
267                 except ImportError:
268                     pass
269                 err += sprintf("\nEnvironment:\n%s\n", pprint.pformat(environ))
270         if _wsgi_server is None:
271             # Serve up the error with our own mini WSGI app.
272             headers = [ ('Content-type', 'text/plain'),
273                         ('Content-length', str(len(err))) ]
274             start_response("500 Internal Server Error", headers)
275             return [err]
276 
277         # Call the normal application
278         return _wsgi_server.wsgi_application(environ, start_response)
279     return application
```

Listing D-2: `nilmrun/processmanager.py`: Process manager for user-provided code and commands. This encapsulates a process and manages aspects of creation, program execution, dynamic code evaluation, and log capture.

Git repository: `https://git.jim.sh/jim/lees/nilmrun.git`
Filename: `nilmrun/processmanager.py`
Revision: `38c3e67cf9f1a9912f94c0f8cad44841d9c48396`

```
#!/usr/bin/python

from nilmdb.utils.printf import *

import threading
import subprocess
import cStringIO
import sys
import os
import signal
import time
import uuid
import psutil
import tempfile
import atexit
import shutil

class ProcessError(Exception):
    pass

class LogReceiver(object):
    """Spawn a thread that listens to a pipe for log messages,
    and stores them locally."""
    def __init__(self, pipe):
        self.pipe = pipe
        self.log = cStringIO.StringIO()
        self.thread = threading.Thread(target = self.run)
        self.thread.start()

    def run(self):
        while True:
            data = os.read(self.pipe, 65536)
            if not data:
                os.close(self.pipe)
                return
            self.log.write(data)

    def getvalue(self):
        return self.log.getvalue()

    def clear(self):
        self.log = cStringIO.StringIO()

class Process(object):
    """Spawn and manage a subprocess, and capture its output."""
    def __init__(self, argv, tempfile = None):
        self.start_time = None

        # Use a pipe for communicating log data
        (rpipe, wpipe) = os.pipe()
        self._log = LogReceiver(rpipe)

        # Stdin is null
        nullfd = os.open(os.devnull, os.O_RDONLY)

        # Spawn the new process
```

```
        try:
            self._process = subprocess.Popen(args = argv, stdin = nullfd,
                                             stdout = wpipe, stderr = wpipe,
                                             close_fds = True, cwd = "/tmp")
        except (OSError, TypeError) as e:
            raise ProcessError(str(e))
        finally:
            # Close the FDs we don't need
            os.close(wpipe)
            os.close(nullfd)

        # Get process info
        self.start_time = time.time()
        self.pid = str(uuid.uuid1(self._process.pid or 0))

    def _join(self, timeout = 1.0):
        start = time.time()
        while True:
            if self._process.poll() is not None:
                return True
            if (time.time() - start) >= timeout:
                return False
            time.sleep(0.1)

    def terminate(self, timeout = 1.0):
        """Terminate a process, and all of its children that are in the same
        process group."""
        try:
            # First give it some time to die on its own
            if self._join(timeout):
                return True

            def getpgid(pid):
                try:
                    return os.getpgid(pid)
                except OSError: # pragma: no cover
                    return None

            def kill(pid, sig):
                try:
                    return os.kill(pid, sig)
                except OSError: # pragma: no cover
                    return

            # Find all children
            group = getpgid(self._process.pid)
            main = psutil.Process(self._process.pid)
            allproc = [ main ] + main.get_children(recursive = True)

            # Kill with SIGTERM, if they're still in this process group
            for proc in allproc:
                if getpgid(proc.pid) == group:
                    kill(proc.pid, signal.SIGTERM)

            # Wait for it to die again
            if self._join(timeout):
                return True

            # One more try with SIGKILL
            for proc in allproc:
                if getpgid(proc.pid) == group:
                    kill(proc.pid, signal.SIGKILL)

            # See if it worked
            return self._join(timeout)
        except psutil.Error: # pragma: no cover (race condition)
            return True
```

```
124
125     def clear_log(self):
126         self._log.clear()
127
128     @property
129     def log(self):
130         return self._log.getvalue()
131
132     @property
133     def alive(self):
134         return self._process.poll() is None
135
136     @property
137     def exitcode(self):
138         return self._process.returncode
139
140     def get_info_prepare(self):
141         """Prepare the process list and measurement for .get_info.
142         Call .get_info() about a second later."""
143         try:
144             main = psutil.Process(self._process.pid)
145             self._process_list = [ main ] + main.get_children(recursive = True)
146             for proc in self._process_list:
147                 proc.get_cpu_percent(0)
148         except psutil.Error: # pragma: no cover (race condition)
149             self._process_list = [ ]
150
151     @staticmethod
152     def get_empty_info():
153         return { "cpu_percent": 0,
154                  "cpu_user": 0,
155                  "cpu_sys": 0,
156                  "mem_phys": 0,
157                  "mem_virt": 0,
158                  "io_read": 0,
159                  "io_write": 0,
160                  "procs": 0 }
161
162     def get_info(self):
163         """Return a dictionary with info about the process CPU and memory
164         usage.  Call .get_info_prepare() about a second before this."""
165         d = self.get_empty_info()
166         for proc in self._process_list:
167             try:
168                 d["cpu_percent"] += proc.get_cpu_percent(0)
169                 cpuinfo = proc.get_cpu_times()
170                 d["cpu_user"] += cpuinfo.user
171                 d["cpu_sys"] += cpuinfo.system
172                 meminfo = proc.get_memory_info()
173                 d["mem_phys"] += meminfo.rss
174                 d["mem_virt"] += meminfo.vms
175                 ioinfo = proc.get_io_counters()
176                 d["io_read"] += ioinfo.read_bytes
177                 d["io_write"] += ioinfo.write_bytes
178                 d["procs"] += 1
179             except psutil.Error:
180                 pass
181         return d
182
183 class ProcessManager(object):
184     """Track and manage a collection of Process objects"""
185     def __init__(self):
186         self.processes = {}
187         self.tmpdirs = {}
188         atexit.register(self._atexit)
189
190     def _cleanup_tmpdir(self, pid):
```

```
        if pid in self.tmpdirs:
            try:
                shutil.rmtree(self.tmpdirs[pid])
            except OSError: # pragma: no cover
                pass
            del self.tmpdirs[pid]

    def _atexit(self):
        # Kill remaining processes, remove their dirs
        for pid in self.processes.keys():
            try:
                self.processes[pid].terminate()
                del self.processes[pid]
                shutil.rmtree(self.tmpdirs[pid])
                del self.tmpdirs[pid]
            except Exception: # pragma: no cover
                pass

    def __iter__(self):
        return iter(self.processes.keys())

    def __getitem__(self, key):
        return self.processes[key]

    def run_code(self, code, args):
        """Evaluate 'code' as if it were placed into a Python file and
        executed.  The arguments, which must be strings, will be
        accessible in the code as sys.argv[1:]."""
        # The easiest way to do this, by far, is to just write the
        # code to a file.  Make a directory to put it in.
        tmpdir = tempfile.mkdtemp(prefix = "nilmrun-usercode-")
        try:
            # Write the code
            codepath = os.path.join(tmpdir, "usercode.py")
            with open(codepath, "w") as f:
                f.write(code)
            # Save the args too, for debugging purposes
            with open(os.path.join(tmpdir, "args.txt"), "w") as f:
                f.write(repr(args))

            # Run the code
            argv = [ sys.executable, "-B", "-s", "-u", codepath ] + args
            pid = self.run_command(argv)

            # Save the temp dir
            self.tmpdirs[pid] = tmpdir
            tmpdir = None  # Don't need to remove it anymore

            return pid
        finally:
            # Clean up tempdir if we didn't finish
            if tmpdir is not None:
                try:
                    shutil.rmtree(tmpdir)
                except OSError: # pragma: no cover
                    pass

    def run_command(self, argv):
        """Execute a command line program"""
        new = Process(argv)
        self.processes[new.pid] = new
        return new.pid

    def terminate(self, pid):
        return self.processes[pid].terminate()

    def remove(self, pid):
```

```
258             self._cleanup_tmpdir(pid)
259             del self.processes[pid]
260 
261         def get_info(self):
262             """Get info about all running PIDs"""
263             info = { "total" : Process.get_empty_info(),
264                      "pids" : {},
265                      "system" : {}
266                      }
267 
268             # Trigger CPU usage collection
269             for pid in self:
270                 self[pid].get_info_prepare()
271             psutil.cpu_percent(0, percpu = True)
272 
273             # Give it some time
274             time.sleep(1)
275 
276             # Retrieve info for system
277             info["system"]["cpu_percent"] = sum(psutil.cpu_percent(0, percpu=True))
278             info["system"]["cpu_max"] = 100.0 * psutil.NUM_CPUS
279             info["system"]["procs"] = len(psutil.get_pid_list())
280             # psutil > 0.6.0's psutil.virtual_memory() would be better here,
281             # but this should give the same info.
282             meminfo = psutil.phymem_usage()
283             info["system"]["mem_total"] = meminfo.total
284             info["system"]["mem_used"] = int(meminfo.total * meminfo.percent / 100)
285 
286             # Retrieve info for each PID
287             for pid in self:
288                 info["pids"][pid] = self[pid].get_info()
289                 # Update totals
290                 for key in info["total"]:
291                     info["total"][key] += info["pids"][pid][key]
292 
293             return info
```

Listing D-3: `nilmrun/testfilter.py`: Filter for validating process management routines. This is used by the test suite.

Git repository: https://git.jim.sh/jim/lees/nilmrun.git
Filename: nilmrun/testfilter.py
Revision: 38c3e67cf9f1a9912f94c0f8cad44841d9c48396

```
1  #!/usr/bin/python
2  
3  from nilmdb.utils.printf import *
4  import time
5  import signal
6  import sys
7  
8  # This is just for testing the process management.
9  def test(n):
10     n = int(n)
11     if n < 0: # raise an exception
12         raise Exception("test exception")
13     if n == 0: # ignore SIGTERM and count to 100
14         n = 100
15         signal.signal(signal.SIGTERM, signal.SIG_IGN)
16     for x in range(n):
17         s = sprintf("dummy %d\n", x)
18         if x & 1:
19             sys.stdout.write(s)
```

```
20          else:
21              sys.stderr.write(s)
22          time.sleep(0.1)
```

D.2 Command Line Tools

Listing D-4: `scripts/nilmrun_server.py (nilmrun-server)`: Script to spawn the standalone NilmRun server. This starts NilmRun using the CherryPy HTTP server, and is typically used for debug or test deployments.

Git repository: https://git.jim.sh/jim/lees/nilmrun.git
Filename: scripts/nilmrun_server.py
Revision: 38c3e67cf9f1a9912f94c0f8cad44841d9c48396

```
1  #!/usr/bin/python
2
3  import nilmrun.server
4  import argparse
5  import os
6  import socket
7
8  def main():
9      """Main entry point for the 'nilmrun-server' command line script"""
10
11     parser = argparse.ArgumentParser(
12         description = 'Run the NilmRun server',
13         formatter_class = argparse.ArgumentDefaultsHelpFormatter,
14         version = nilmrun.__version__)
15
16     group = parser.add_argument_group("Standard options")
17     group.add_argument('-a', '--address',
18                        help = 'Only listen on the given address',
19                        default = '0.0.0.0')
20     group.add_argument('-p', '--port', help = 'Listen on the given port',
21                        type = int, default = 12381)
22     group.add_argument('-q', '--quiet', help = 'Silence output',
23                        action = 'store_true')
24     group.add_argument('-t', '--traceback',
25                        help = 'Provide tracebacks in client errors',
26                        action = 'store_true', default = False)
27
28     args = parser.parse_args()
29
30     # Configure the server
31     if args.quiet:
32         embedded = True
33     else:
34         embedded = False
35     server = nilmrun.server.Server(host = args.address,
36                                    port = args.port,
37                                    embedded = embedded,
38                                    force_traceback = args.traceback)
39
40     # Print info
41     if not args.quiet:
42         print "NilmRun version: %s" % nilmrun.__version__
43         print ("Note: This server does not do any authentication! " +
44                "Anyone who can connect can run arbitrary commands.")
```

```
        if args.address == '0.0.0.0' or args.address == '::':
            host = socket.getfqdn()
        else:
            host = args.address
        print "Server URL: http://%s:%d/" % ( host, args.port)
        print "----"

    server.start(blocking = True)

    if not args.quiet:
        print "Shutting down"

if __name__ == "__main__":
    main()
```

Listing D-5: `scripts/ps.py (nilmrun-ps)`: List running processes.

Git repository: https://git.jim.sh/jim/lees/nilmrun.git
Filename: scripts/ps.py
Revision: 38c3e67cf9f1a9912f94c0f8cad44841d9c48396

```
#!/usr/bin/python

from nilmdb.client.httpclient import HTTPClient, ClientError, ServerError
from nilmdb.utils.printf import *
from nilmdb.utils import datetime_tz
import nilmrun

import argparse
import os

def main():
    """List NilmRun processes"""
    def_url = os.environ.get("NILMRUN_URL", "http://localhost/nilmrun/")
    parser = argparse.ArgumentParser(
        description = 'List NilmRun processes',
        formatter_class = argparse.ArgumentDefaultsHelpFormatter,
        version = nilmrun.__version__)
    group = parser.add_argument_group("Standard options")
    group.add_argument('-u', '--url',
                       help = 'NilmRun server URL', default = def_url)
    group.add_argument('-n', '--noverify', action="store_true",
                       help = 'Disable SSL certificate verification')
    args = parser.parse_args()

    client = HTTPClient(baseurl = args.url, verify_ssl = not args.noverify)
    # Print overall system info
    info = client.get("process/info")
    total = info['total']
    system = info['system']
    printf(" procs: %d nilm, %d other\n", info['total']['procs'],
           info['system']['procs'] - info['total']['procs'])
    printf("   cpu: %d%% nilm, %d%% other, %d%% max\n",
           round(info['total']['cpu_percent']),
           round(info['system']['cpu_percent'] - info['total']['cpu_percent']),
           round(info['system']['cpu_max']))
    printf("   mem: %d MiB used, %d MiB total, %d%%\n",
           round(info['system']['mem_used'] / 1048576.0),
           round(info['system']['mem_total'] / 1048576.0),
           round(info['system']['mem_used'] * 100.0
                 / info['system']['mem_total']))

    # Print process detail for each managed process
    fmt = "%-36s %-6s %-15s %-4s %-3s %-5s\n"
```

```
    printf(fmt, "PID", "STATE", "SINCE", "PROC", "CPU", "LOG")

    if len(info['pids']) == 0:
        printf("No running processes\n")
        raise SystemExit(0)

    for pid in sorted(info['pids'].keys()):
        pidinfo = client.get("process/status", { "pid": pid })
        if pidinfo['alive']:
            status = "alive"
        else:
            if pidinfo['exitcode']:
                status = "error"
            else:
                status = "done"
        dt = datetime_tz.datetime_tz.fromtimestamp(pidinfo['start_time'])
        since = dt.strftime("%m/%d-%H:%M:%S")
        printf(fmt, pid, status, since, info['pids'][pid]['procs'],
               str(int(round(info['pids'][pid]['cpu_percent']))),
               len(pidinfo['log']))

if __name__ == "__main__":
    main()
```

Listing D-6: `scripts/run.py (nilmrun-run)`: Run a command on a NilmRun server. Optionally supports waiting for the process to exit while displaying its output.

Git repository: https://git.jim.sh/jim/lees/nilmrun.git
Filename: scripts/run.py
Revision: 38c3e67cf9f1a9912f94c0f8cad44841d9c48396

```
#!/usr/bin/python

from nilmdb.client.httpclient import HTTPClient, ClientError, ServerError
from nilmdb.utils.printf import *
import nilmrun

import argparse
import os
import time
import sys

def main():
    """Run a command on the NilmRun server"""
    def_url = os.environ.get("NILMRUN_URL", "http://localhost/nilmrun/")
    parser = argparse.ArgumentParser(
        description = 'Run a command on the NilmRun server',
        formatter_class = argparse.ArgumentDefaultsHelpFormatter,
        version = nilmrun.__version__)
    group = parser.add_argument_group("Standard options")
    group.add_argument('-u', '--url',
                       help = 'NilmRun server URL', default = def_url)
    group.add_argument('-n', '--noverify', action="store_true",
                       help = 'Disable SSL certificate verification')
    group = parser.add_argument_group("Program")
    group.add_argument('-d', '--detach', action="store_true",
                       help = 'Run process and return immediately without '
                       'printing its output')
    group.add_argument('cmd', help="Command to run")
    group.add_argument('arg', nargs=argparse.REMAINDER,
                       help="Arguments for command")
    args = parser.parse_args()
```

```
32
33      client = HTTPClient(baseurl = args.url, verify_ssl = not args.noverify)
34
35      # Run command
36      pid = client.post("run/command", { "argv": [ args.cmd ] + args.arg })
37
38      # If we're detaching, just print the PID
39      if args.detach:
40          print pid
41          raise SystemExit(0)
42
43      # Otherwise, watch the log output, and kill the process when it's done
44      # or when this script terminates.
45      try:
46          while True:
47              s = client.get("process/status", { "pid": pid, "clear": 1 })
48              sys.stdout.write(s['log'])
49              sys.stdout.flush()
50              if not s['alive']:
51                  break
52              time.sleep(1)
53      finally:
54          s = client.post("process/remove", { "pid": pid })
55
56      raise SystemExit(s['exitcode'])
57
58  if __name__ == "__main__":
59      main()
```

Listing D-7: `scripts/kill.py (nilmrun-kill)`: Kill or remove a process. Removes the process from the NilmRun manager and retrieves its final output.

Git repository: https://git.jim.sh/jim/lees/nilmrun.git
Filename: scripts/kill.py
Revision: 38c3e67cf9f1a9912f94c0f8cad44841d9c48396

```
1   #!/usr/bin/python
2
3   from nilmdb.client.httpclient import HTTPClient, ClientError, ServerError
4   from nilmdb.utils.printf import *
5   import nilmrun
6
7   import argparse
8   import os
9   import sys
10
11  def main():
12      """Kill/remove a process from the NilmRun server"""
13      def_url = os.environ.get("NILMRUN_URL", "http://localhost/nilmrun/")
14      parser = argparse.ArgumentParser(
15          description = 'Kill/remove a process from the NilmRun server',
16          formatter_class = argparse.ArgumentDefaultsHelpFormatter,
17          version = nilmrun.__version__)
18      group = parser.add_argument_group("Standard options")
19      group.add_argument('-u', '--url',
20                         help = 'NilmRun server URL', default = def_url)
21      group.add_argument('-n', '--noverify', action="store_true",
22                         help = 'Disable SSL certificate verification')
23      group = parser.add_argument_group("Program")
24      group.add_argument('-q', '--quiet', action="store_true",
25                         help = "Don't print out the final log contents")
26      group.add_argument('pid', nargs='+', help="PIDs to kill")
```

```
    args = parser.parse_args()

    client = HTTPClient(baseurl = args.url, verify_ssl = not args.noverify)

    # Kill or remove process
    all_failed = True
    for pid in args.pid:
        try:
            s = client.post("process/remove", { "pid": pid })
            if not args.quiet:
                sys.stdout.write(s['log'])
            all_failed = False
        except ClientError as e:
            if "404" in e.status:
                fprintf(sys.stderr, "no such pid: %s\n", pid)
            else:
                raise

    # Return error if we failed to remove any of them
    if all_failed:
        raise SystemExit(1)

if __name__ == "__main__":
    main()
```

D.3 Test Suite

Listing D-8: `tests/test_nilmrun.py`: Test the NilmRun server and process management.

Git repository: https://git.jim.sh/jim/lees/nilmrun.git
Filename: tests/test_nilmrun.py
Revision: 38c3e67cf9f1a9912f94c0f8cad44841d9c48396

```
# -*- coding: utf-8 -*-

import nilmrun.server

from nilmdb.client.httpclient import HTTPClient, ClientError, ServerError

from nilmdb.utils.printf import *

from nose.plugins.skip import SkipTest
from nose.tools import *
from nose.tools import assert_raises

import itertools
import distutils.version
import os
import sys
import threading
import cStringIO
import simplejson as json
import unittest
import warnings
import time
import re
import urllib2
from urllib2 import urlopen, HTTPError
import requests
```

```
import pprint
import textwrap

from testutil.helpers import *

testurl = "http://localhost:32181/"
#testurl = "http://bucket.mit.edu/nilmrun/"

def setup_module():
    global test_server

    # Start web app on a custom port
    test_server = nilmrun.server.Server(host = "127.0.0.1",
                                        port = 32181,
                                        force_traceback = True)
    test_server.start(blocking = False)

def teardown_module():
    global test_server
    # Close web app
    test_server.stop()

class TestClient(object):

    def wait_kill(self, client, pid, timeout = 1):
        time.sleep(timeout)
        status = client.get("process/status", { "pid": pid })
        if not status["alive"]:
            raise AssertionError("died before we could kill it")
        status = client.post("process/remove", { "pid": pid })
        if status["alive"]:
            raise AssertionError("didn't get killed")
        return status

    def wait_end(self, client, pid, timeout = 5, remove = True):
        start = time.time()
        status = None
        while (time.time() - start) < timeout:
            status = client.get("process/status", { "pid": pid })
            if status["alive"] == False:
                break
            time.sleep(0.1)
        else:
            raise AssertionError("process " + str(pid) + " didn't die in " +
                                 str(timeout) + " seconds: " + repr(status))
        if remove:
            status = client.post("process/remove", { "pid": pid })
        return status

    def test_client_01_basic(self):
        client = HTTPClient(baseurl = testurl)
        version = client.get("version")
        eq_(distutils.version.LooseVersion(version),
            distutils.version.LooseVersion(nilmrun.__version__))

        in_("This is NilmRun", client.get(""))

        with assert_raises(ClientError):
            client.get("favicon.ico")

    def test_client_02_manager(self):
        client = HTTPClient(baseurl = testurl)

        eq_(client.get("process/list"), [])

        with assert_raises(ClientError) as e:
            client.get("process/status", { "pid": 12345 })
```

```
94          in_("No such PID", str(e.exception))
95          with assert_raises(ClientError):
96              client.get("process/remove", { "pid": 12345 })
97          in_("No such PID", str(e.exception))
98
99      def test_client_03_run_command(self):
100         client = HTTPClient(baseurl = testurl, post_json = True)
101         eq_(client.get("process/list"), [])
102
103         def do(argv, kill):
104             pid = client.post("run/command", { "argv": argv } )
105             eq_(client.get("process/list"), [pid])
106             if kill:
107                 return self.wait_kill(client, pid)
108             return self.wait_end(client, pid)
109
110         # Simple command
111         status = do(["pwd"], False)
112         eq_(status["exitcode"], 0)
113         eq_("/tmp\n", status["log"])
114
115         # Command with args
116         status = do(["expr", "1", "+", "2"], False)
117         eq_(status["exitcode"], 0)
118         eq_("3\n", status["log"])
119
120         # Missing command
121         with assert_raises(ClientError) as e:
122             do(["/no-such-command-blah-blah"], False)
123         in_("No such file or directory", str(e.exception))
124
125         # Kill a slow command
126         status = do(["sleep", "60"], True)
127         ne_(status["exitcode"], 0)
128
129     def _run_testfilter(self, client, args):
130         code = textwrap.dedent("""
131         import nilmrun.testfilter
132         import simplejson as json
133         import sys
134         nilmrun.testfilter.test(json.loads(sys.argv[1]))
135         """)
136         jsonargs = json.dumps(args)
137         return client.post("run/code", { "code": code, "args": [ jsonargs ] })
138
139     def test_client_04_process_basic(self):
140         client = HTTPClient(baseurl = testurl, post_json = True)
141
142         # start dummy filter
143         pid = self._run_testfilter(client, 30)
144         eq_(client.get("process/list"), [pid])
145         time.sleep(1)
146
147         # Verify that status looks OK
148         status = client.get("process/status", { "pid": pid, "clear": True })
149         for x in [ "pid", "alive", "exitcode", "start_time", "log" ]:
150             in_(x, status)
151         in_("dummy 0\ndummy 1\ndummy 2\ndummy 3\n", status["log"])
152         eq_(status["alive"], True)
153         eq_(status["exitcode"], None)
154
155         # Check that the log got cleared
156         status = client.get("process/status", { "pid": pid })
157         nin_("dummy 0\ndummy 1\ndummy 2\ndummy 3\n", status["log"])
158
159         # See that it ended properly
160         status = self.wait_end(client, pid, remove = False)
```

```
        in_("dummy 27\ndummy 28\ndummy 29\n", status["log"])
        eq_(status["exitcode"], 0)

        # Remove it
        killstatus = client.post("process/remove", { "pid": pid })
        eq_(status, killstatus)
        eq_(client.get("process/list"), [])
        with assert_raises(ClientError) as e:
            client.post("process/remove", { "pid": pid })
        in_("No such PID", str(e.exception))

    def test_client_05_process_terminate(self):
        client = HTTPClient(baseurl = testurl, post_json = True)

        # Trigger exception in filter
        pid = self._run_testfilter(client, -1)
        time.sleep(0.5)
        status = client.get("process/status", { "pid": pid })
        eq_(status["alive"], False)
        eq_(status["exitcode"], 1)
        in_("Exception: test exception", status["log"])
        client.post("process/remove", { "pid": pid })

        # Kill a running filter by removing it early
        newpid = self._run_testfilter(client, 50)
        ne_(newpid, pid)
        time.sleep(0.5)
        start = time.time()
        status = client.post("process/remove", { "pid": newpid })
        elapsed = time.time() - start
        # Should have died in slightly over 1 second
        assert(0.5 < elapsed < 2)
        eq_(status["alive"], False)
        ne_(status["exitcode"], 0)

        # No more
        eq_(client.get("process/list"), [])

        # Try to remove a running filter that ignored SIGTERM
        pid = self._run_testfilter(client, 0)
        start = time.time()
        status = client.post("process/remove", { "pid": pid })
        elapsed = time.time() - start
        # Should have died in slightly over 2 seconds
        assert(1.5 < elapsed < 3)
        eq_(status["alive"], False)
        ne_(status["exitcode"], 0)

    @unittest.skip("needs a running nilmdb; trainola moved to nilmtools")
    def test_client_06_trainola(self):
        client = HTTPClient(baseurl = testurl, post_json = True)
        data = { "url": "http://bucket.mit.edu/nilmdb",
                 "dest_stream": "/sharon/prep-a-matches",
                 "stream": "/sharon/prep-a",
                 "start": 1366111383280463,
                 "end": 1366126163457797,
                 "columns": [ { "name": "P1", "index": 0 },
                              { "name": "Q1", "index": 1 },
                              { "name": "P3", "index": 2 } ],
                 "exemplars": [
                     { "name": "Boiler Pump ON",
                       "url": "http://bucket.mit.edu/nilmdb",
                       "stream": "/sharon/prep-a",
                       "start": 1366260494269078,
                       "end": 1366260608185031,
                       "dest_column": 0,
                       "columns": [ { "name": "P1", "index": 0 },
```

```
                                    { "name": "Q1", "index": 1 }
                                    ]
                      },
                    { "name": "Boiler Pump OFF",
                      "url": "http://bucket.mit.edu/nilmdb",
                      "stream": "/sharon/prep-a",
                      "start": 1366260864215764,
                      "end": 1366260870882998,
                      "dest_column": 1,
                      "columns": [ { "name": "P1", "index": 0 },
                                   { "name": "Q1", "index": 1 }
                                   ]
                      }
                    ]
                }
        pid = client.post("run/code", { "code": "import nilmtools.trainola\n" +
                                        "nilmtools.trainola.main()",
                                        "args": [ data ] })
        while True:
            status = client.get("process/status", { "pid": pid, "clear": 1 })
            sys.stdout.write(status["log"])
            sys.stdout.flush()
            if status["alive"] == False:
                break
            time.sleep(0.1)
        status = client.post("process/remove", { "pid": pid })
        os._exit(int(status["exitcode"]))

    def test_client_07_run_code(self):
        client = HTTPClient(baseurl = testurl, post_json = True)
        eq_(client.get("process/list"), [])

        def do(code, args, kill):
            if args is not None:
                pid = client.post("run/code", { "code": code, "args": args } )
            else:
                pid = client.post("run/code", { "code": code } )
            eq_(client.get("process/list"), [pid])
            if kill:
                return self.wait_kill(client, pid)
            return self.wait_end(client, pid)

        # basic code snippet
        code = textwrap.dedent("""
        print 'hello'
        def foo(arg):
            print 'world'
        """)
        status = do(code, [], False)
        eq_("hello\n", status["log"])
        eq_(status["exitcode"], 0)

        # compile error
        code = textwrap.dedent("""
        def foo(arg:
            print 'hello'
        """)
        status = do(code, [], False)
        in_("SyntaxError", status["log"])
        eq_(status["exitcode"], 1)

        # traceback in user code should be formatted nicely
        code = textwrap.dedent("""
        def foo(arg):
            raise Exception(arg)
        foo(123)
        """)
```

```
        status = do(code, [], False)
        cleaned_log = re.sub('File "[^"]*",', 'File "",', status["log"])
        eq_('Traceback (most recent call last):\n' +
        '  File "", line 4, in <module>\n' +
        '    foo(123)\n' +
        '  File "", line 3, in foo\n' +
        '    raise Exception(arg)\n' +
        'Exception: 123\n', cleaned_log)
        eq_(status["exitcode"], 1)

        # argument handling (strings come in as unicode)
        code = textwrap.dedent("""
        import sys
        print sys.argv[1].encode('ascii'), sys.argv[2]
        sys.exit(0)  # also test raising SystemExit
        """)
        with assert_raises(ClientError) as e:
        do(code, ["hello", 123], False)
        in_("400 Bad Request", str(e.exception))
        status = do(code, ["hello", "123"], False)
        eq_(status["log"], "hello 123\n")
        eq_(status["exitcode"], 0)

        # try killing a long-running process
        code = textwrap.dedent("""
        import time
        print 'hello'
        time.sleep(60)
        print 'world'
        """)
        status = do(code, [], True)
        eq_(status["log"], "hello\n")
        ne_(status["exitcode"], 0)

        # default arguments are empty
        code = textwrap.dedent("""
        import sys
        print 'args:', len(sys.argv[1:])
        """)
        status = do(code, None, False)
        eq_(status["log"], "args: 0\n")
        eq_(status["exitcode"], 0)

    def test_client_08_bad_types(self):
        client = HTTPClient(baseurl = testurl, post_json = True)

        with assert_raises(ClientError) as e:
            client.post("run/code", { "code": "asdf", "args": "qwer" })
        in_("must be a list", str(e.exception))

        with assert_raises(ClientError) as e:
            client.post("run/command", { "argv": "asdf" })
        in_("must be a list", str(e.exception))

    def test_client_09_info(self):
        client = HTTPClient(baseurl = testurl, post_json = True)

        # start some processes
        a = client.post("run/command", { "argv": ["sleep","60"] } )
        b = client.post("run/command", { "argv": ["sh","-c","sleep 2;true"] } )
        c = client.post("run/command", { "argv": [
            "sh","-c","dd if=/dev/zero of=/dev/null;true"] } )
        d = client.post("run/command", { "argv": [
            "dd", "if=/dev/zero", "of=/dev/null" ] } )

        info = client.get("process/info")
        eq_(info["pids"][a]["procs"], 1)
```

```
362         eq_(info["pids"][b]["procs"], 2)
363         eq_(info["pids"][c]["procs"], 2)
364         eq_(info["pids"][d]["procs"], 1)
365         eq_(info["total"]["procs"], 6)
366         lt_(info["pids"][a]["cpu_percent"], 50)
367         lt_(20, info["pids"][c]["cpu_percent"])
368         lt_(80, info["system"]["cpu_percent"])
369
370         for x in range(10):
371             time.sleep(1)
372             info = client.get("process/info")
373             if info["pids"][b]["procs"] != 2:
374                 break
375         else:
376             raise Exception("process B didn't die: " + str(info["pids"][b]))
377
378         # kill all processes
379         for pid in client.get("process/list"):
380             client.post("process/remove", { "pid": pid })
381
382     def test_client_10_unicode(self):
383         client = HTTPClient(baseurl = testurl, post_json = True)
384         eq_(client.get("process/list"), [])
385         def verify(cmd, result):
386             pid = client.post("run/command", { "argv": [
387                 "/bin/bash", "-c", cmd ] })
388             eq_(client.get("process/list"), [pid])
389             status = self.wait_end(client, pid)
390             eq_(result, status["log"])
391
392         # Unicode should work
393         verify("echo -n hello", "hello")
394         verify(u"echo -n ☠", u"☠")
395         verify("echo -ne \\\\xe2\\\\x98\\\\xa0", u"☠")
396
397         # Programs that spit out invalid UTF-8 should get replacement
398         # markers
399         verify("echo -ne \\\\xae", u"\ufffd")
400
401     def test_client_11_atexit(self):
402         # Leave a directory and running process behind, for the atexit
403         # handler to clean up.  Here we trigger the atexit manually,
404         # since it's hard to trigger it as part of the test suite.
405         client = HTTPClient(baseurl = testurl, post_json = True)
406         code = textwrap.dedent("""
407         import time
408         time.sleep(10)
409         """)
410         client.post("run/code", { "code": code, "args": [ "hello"] })
411
412         # Trigger atexit function
413         test_server._manager._atexit()
414
415         # Ensure no processes exit
416         eq_(client.get("process/list"), [])
```

Appendix E

NilmTools Implementation

This appendix contains the implementation of the NilmTools suite of filters and tools.

E.1 Libraries

Listing E-1: `nilmtools/filter.py`: Base framework for NilmDB filters and filter-like tools. This class manages basic command-line interface parsing, automatic interval selection, metadata management, and retrieval and insertion of data.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/filter.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1  #!/usr/bin/python
2
3  from __future__ import absolute_import
4
5  import nilmdb.client
6  from nilmdb.client import Client
7  from nilmdb.client.numpyclient import NumpyClient
8  from nilmdb.utils.printf import *
9  from nilmdb.utils.time import (parse_time, timestamp_to_human,
10                                 timestamp_to_seconds)
11 from nilmdb.utils.interval import Interval
12
13 import nilmtools
14
15 import itertools
16 import time
17 import sys
18 import re
19 import argparse
20 import numpy as np
21 import cStringIO
22 import functools
23
```

```
class ArgumentError(Exception):
    pass

class MissingDestination(Exception):
    def __init__(self, args, src, dest):
        self.parsed_args = args
        self.src = src
        self.dest = dest
        Exception.__init__(self, "destination path " + dest.path + " not found")

class StreamInfo(object):
    def __init__(self, url, info):
        self.url = url
        self.info = info
        try:
            self.path = info[0]
            self.layout = info[1]
            self.layout_type = self.layout.split('_')[0]
            self.layout_count = int(self.layout.split('_')[1])
            self.total_count = self.layout_count + 1
            self.timestamp_min = info[2]
            self.timestamp_max = info[3]
            self.rows = info[4]
            self.seconds = nilmdb.utils.time.timestamp_to_seconds(info[5])
        except IndexError, TypeError:
            pass

    def string(self, interhost):
        """Return stream info as a string.  If interhost is true,
        include the host URL."""
        if interhost:
            return sprintf("[%s] ", self.url) + str(self)
        return str(self)

    def __str__(self):
        """Return stream info as a string."""
        return sprintf("%s (%s), %.2fM rows, %.2f hours",
                       self.path, self.layout, self.rows / 1e6,
                       self.seconds / 3600.0)

def get_stream_info(client, path):
    """Return a StreamInfo object about the given path, or None if it
    doesn't exist"""
    streams = client.stream_list(path, extended = True)
    if len(streams) != 1:
        return None
    return StreamInfo(client.geturl(), streams[0])

# Filter processing for a single interval of data.
def process_numpy_interval(interval, extractor, inserter, warn_rows,
                           function, args = None):
    """For the given 'interval' of data, extract data, process it
    through 'function', and insert the result.

    'extractor' should be a function like NumpyClient.stream_extract_numpy
    but with the the interval 'start' and 'end' as the only parameters,
    e.g.:
       extractor = functools.partial(NumpyClient.stream_extract_numpy,
                                     src_path, layout = l, maxrows = m)

    'inserter' should be a function like NumpyClient.stream_insert_context
    but with the interval 'start' and 'end' as the only parameters, e.g.:
       inserter = functools.partial(NumpyClient.stream_insert_context,
                                    dest_path)

    If 'warn_rows' is not None, print a warning to stdout when the
    number of unprocessed rows exceeds this amount.
```

```
91
92      See process_numpy for details on 'function' and 'args'.
93      """
94      if args is None:
95          args = []
96
97      with inserter(interval.start, interval.end) as insert_ctx:
98          insert_func = insert_ctx.insert
99          old_array = np.array([])
100         for new_array in extractor(interval.start, interval.end):
101             # If we still had old data left, combine it
102             if old_array.shape[0] != 0:
103                 array = np.vstack((old_array, new_array))
104             else:
105                 array = new_array
106
107             # Pass the data to the user provided function
108             processed = function(array, interval, args, insert_func, False)
109
110             # Send any pending data that the user function inserted
111             insert_ctx.send()
112
113             # Save the unprocessed parts
114             if processed >= 0:
115                 old_array = array[processed:]
116             else:
117                 raise Exception(
118                     sprintf("%s return value %s must be >= 0",
119                             str(function), str(processed)))
120
121             # Warn if there's too much data remaining
122             if warn_rows is not None and old_array.shape[0] > warn_rows:
123                 printf("warning: %d unprocessed rows in buffer\n",
124                        old_array.shape[0])
125
126         # Last call for this contiguous interval
127         if old_array.shape[0] != 0:
128             processed = function(old_array, interval, args,
129                                  insert_func, True)
130             if processed != old_array.shape[0]:
131                 # Truncate the interval we're inserting at the first
132                 # unprocessed data point.  This ensures that
133                 # we'll not miss any data when we run again later.
134                 insert_ctx.update_end(old_array[processed][0])
135
136 def example_callback_function(data, interval, args, insert_func, final):
137     """Example of the signature for the function that gets passed
138     to process_numpy_interval.
139
140     'data': array of data to process -- may be empty
141
142     'interval': overall interval we're processing (but not necessarily
143     the interval of this particular chunk of data)
144
145     'args': opaque arguments passed to process_numpy
146
147     'insert_func': function to call in order to insert array of data.
148     Should be passed a 2-dimensional array of data to insert.
149     Data timestamps must be within the provided interval.
150
151     'final': True if this is the last bit of data for this
152     contiguous interval, False otherwise.
153
154     Return value of 'function' is the number of data rows processed.
155     Unprocessed data will be provided again in a subsequent call
156     (unless 'final' is True).
157
```

```
158         If unprocessed data remains after 'final' is True, the interval
159         being inserted will be ended at the timestamp of the first
160         unprocessed data point.
161         """
162         raise NotImplementedError("example_callback_function does nothing")
163 
164     class Filter(object):
165 
166         def __init__(self, parser_description = None):
167             self._parser = None
168             self._client_src = None
169             self._client_dest = None
170             self._using_client = False
171             self.src = None
172             self.dest = None
173             self.start = None
174             self.end = None
175             self._interhost = False
176             self._force_metadata = False
177             if parser_description is not None:
178                 self.setup_parser(parser_description)
179                 self.parse_args()
180 
181         @property
182         def client_src(self):
183             if self._using_client:
184                 raise Exception("Filter client is in use; make another")
185             return self._client_src
186 
187         @property
188         def client_dest(self):
189             if self._using_client:
190                 raise Exception("Filter client is in use; make another")
191             return self._client_dest
192 
193         def setup_parser(self, description = "Filter data", skip_paths = False):
194             parser = argparse.ArgumentParser(
195                 formatter_class = argparse.RawDescriptionHelpFormatter,
196                 version = nilmtools.__version__,
197                 description = description)
198             group = parser.add_argument_group("General filter arguments")
199             group.add_argument("-u", "--url", action="store",
200                                default="http://localhost/nilmdb/",
201                                help="Server URL (default: %(default)s)")
202             group.add_argument("-U", "--dest-url", action="store",
203                                help="Destination server URL "
204                                "(default: same as source)")
205             group.add_argument("-D", "--dry-run", action="store_true",
206                                default = False,
207                                help="Just print intervals that would be "
208                                "processed")
209             group.add_argument("-F", "--force-metadata", action="store_true",
210                                default = False,
211                                help="Force metadata changes if the dest "
212                                "doesn't match")
213             group.add_argument("-s", "--start",
214                                metavar="TIME", type=self.arg_time,
215                                help="Starting timestamp for intervals "
216                                "(free-form, inclusive)")
217             group.add_argument("-e", "--end",
218                                metavar="TIME", type=self.arg_time,
219                                help="Ending timestamp for intervals "
220                                "(free-form, noninclusive)")
221             if not skip_paths:
222                 # Individual filter scripts might want to add these arguments
223                 # themselves, to include multiple sources in a different order
224                 # (for example).  "srcpath" and "destpath" arguments must exist,
```

```
225             # though.
226             group.add_argument("srcpath", action="store",
227                                help="Path of source stream, e.g. /foo/bar")
228             group.add_argument("destpath", action="store",
229                                help="Path of destination stream, e.g. /foo/bar")
230         self._parser = parser
231         return parser
232 
233     def set_args(self, url, dest_url, srcpath, destpath, start, end,
234                  parsed_args = None, quiet = True):
235         """Set arguments directly from parameters"""
236         if dest_url is None:
237             dest_url = url
238         if url != dest_url:
239             self._interhost = True
240 
241         self._client_src = Client(url)
242         self._client_dest = Client(dest_url)
243 
244         if (not self._interhost) and (srcpath == destpath):
245             raise ArgumentError("source and destination path must be different")
246 
247         # Open the streams
248         self.src = get_stream_info(self._client_src, srcpath)
249         if not self.src:
250             raise ArgumentError("source path " + srcpath + " not found")
251 
252         self.dest = get_stream_info(self._client_dest, destpath)
253         if not self.dest:
254             raise MissingDestination(parsed_args, self.src,
255                                      StreamInfo(dest_url, [destpath]))
256 
257         self.start = start
258         self.end = end
259 
260         # Print info
261         if not quiet:
262             print "Source:", self.src.string(self._interhost)
263             print "  Dest:", self.dest.string(self._interhost)
264 
265     def parse_args(self, argv = None):
266         """Parse arguments from a command line"""
267         args = self._parser.parse_args(argv)
268 
269         self.set_args(args.url, args.dest_url, args.srcpath, args.destpath,
270                       args.start, args.end, quiet = False, parsed_args = args)
271 
272         self._force_metadata = args.force_metadata
273         if args.dry_run:
274             for interval in self.intervals():
275                 print interval.human_string()
276             raise SystemExit(0)
277         return args
278 
279     def intervals(self):
280         """Generate all the intervals that this filter should process"""
281         self._using_client = True
282 
283         if self._interhost:
284             # Do the difference ourselves
285             s_intervals = ( Interval(start, end)
286                             for (start, end) in
287                             self._client_src.stream_intervals(
288                                 self.src.path,
289                                 start = self.start, end = self.end) )
290             d_intervals = ( Interval(start, end)
291                             for (start, end) in
```

```
                            self._client_dest.stream_intervals(
                                self.dest.path,
                                start = self.start, end = self.end) )
            intervals = nilmdb.utils.interval.set_difference(s_intervals,
                                                             d_intervals)
        else:
            # Let the server do the difference for us
            intervals = ( Interval(start, end)
                          for (start, end) in
                          self._client_src.stream_intervals(
                              self.src.path, diffpath = self.dest.path,
                              start = self.start, end = self.end) )
        # Optimize intervals: join intervals that are adjacent
        for interval in nilmdb.utils.interval.optimize(intervals):
            yield interval
        self._using_client = False

    # Misc helpers
    @staticmethod
    def arg_time(toparse):
        """Parse a time string argument"""
        try:
            return nilmdb.utils.time.parse_time(toparse)
        except ValueError as e:
            raise argparse.ArgumentTypeError(sprintf("%s \"%s\"",
                                                     str(e), toparse))

    def check_dest_metadata(self, data):
        """See if the metadata jives, and complain if it doesn't.  For
        each key in data, if the stream contains the key, it must match
        values.  If the stream does not contain the key, it is created."""
        metadata = self._client_dest.stream_get_metadata(self.dest.path)
        if not self._force_metadata:
            for key in data:
                wanted = data[key]
                if not isinstance(wanted, basestring):
                    wanted = str(wanted)
                val = metadata.get(key, wanted)
                # Force UTF-8 encoding for comparison and display
                wanted = wanted.encode('utf-8')
                val = val.encode('utf-8')
                key = key.encode('utf-8')
                if val != wanted and self.dest.rows > 0:
                    m =  "Metadata in destination stream:\n"
                    m += "  %s = %s\n" % (key, val)
                    m += "doesn't match desired data:\n"
                    m += "  %s = %s\n" % (key, wanted)
                    m += "Refusing to change it.  To prevent this error, "
                    m += "change or delete the metadata with nilmtool,\n"
                    m += "remove existing data from the stream, or "
                    m += "retry with --force-metadata."
                    raise Exception(m)
        # All good -- write the metadata in case it's not already there
        self._client_dest.stream_update_metadata(self.dest.path, data)

    # The main filter processing method.
    def process_numpy(self, function, args = None, rows = 100000,
                      intervals = None):
        """Calls process_numpy_interval for each interval that currently
        exists in self.src, but doesn't exist in self.dest.  It will
        process the data in chunks as follows:

        For each chunk of data, call 'function' with a Numpy array
        corresponding to the data.  The data is converted to a Numpy
        array in chunks of 'rows' rows at a time.

        If 'intervals' is not None, process those intervals instead of
```

```
359         the default list.
360
361         'function' should be defined with the same interface as
362         nilmtools.filter.example_callback_function.  See the
363         documentation of that for details.  'args' are passed to
364         'function'.
365         """
366         extractor = NumpyClient(self.src.url).stream_extract_numpy
367         inserter = NumpyClient(self.dest.url).stream_insert_numpy_context
368
369         extractor_func = functools.partial(extractor, self.src.path,
370                                            layout = self.src.layout,
371                                            maxrows = rows)
372         inserter_func = functools.partial(inserter, self.dest.path)
373
374         for interval in (intervals or self.intervals()):
375             print "Processing", interval.human_string()
376             process_numpy_interval(interval, extractor_func, inserter_func,
377                                    rows * 3, function, args)
378
379 def main(argv = None):
380     # This is just a dummy function; actual filters can use the other
381     # functions to prepare stuff, and then do something with the data.
382     f = Filter()
383     parser = f.setup_parser()
384     args = f.parse_args(argv)
385     for i in f.intervals():
386         print "Generic filter: need to handle", i.human_string()
387
388 if __name__ == "__main__":
389     main()
```

Listing E-2: `nilmtools/math.py`: Useful mathematical functions for NILM usage. Provides miscellaneous functions and routines used by other filters and code.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/math.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1  #!/usr/bin/python
2
3  # Miscellaenous useful mathematical functions
4  from nilmdb.utils.printf import *
5  from numpy import *
6  from scipy import *
7
8  def sfit4(data, fs):
9      """(A, f0, phi, C) = sfit4(data, fs)
10
11     Compute 4-parameter (unknown-frequency) least-squares fit to
12     sine-wave data, according to IEEE Std 1241-2010 Annex B
13
14     Input:
15       data  vector of input samples
16       fs    sampling rate (Hz)
17
18     Output:
19       Parameters [A, f0,  phi, C] to fit the equation
20         x[n] = A * sin(f0/fs * 2 * pi * n + phi) + C
21       where n is sample number.  Or, as a function of time:
22         x(t) = A * sin(f0 * 2 * pi * t + phi) + C
23
```

```
    by Jim Paris
    (Verified to match sfit4.m)
    """
    N = len(data)
    t = linspace(0, (N-1) / float(fs), N)

    ## Estimate frequency using FFT (step b)
    Fc = fft(data)
    F = abs(Fc)
    F[0] = 0   # eliminate DC

    # Find pair of spectral lines with largest amplitude:
    # resulting values are in F(i) and F(i+1)
    i = argmax(F[0:int(N/2)] + F[1:int(N/2+1)])

    # Interpolate FFT to get a better result (from Markus [B37])
    try:
        U1 = real(Fc[i])
        U2 = real(Fc[i+1])
        V1 = imag(Fc[i])
        V2 = imag(Fc[i+1])
        n = 2 * pi / N
        ni1 = n * i
        ni2 = n * (i+1)
        K = ((V2-V1)*sin(ni1) + (U2-U1)*cos(ni1)) / (U2-U1)
        Z1 = V1 * (K - cos(ni1)) / sin(ni1) + U1
        Z2 = V2 * (K - cos(ni2)) / sin(ni2) + U2
        i = arccos((Z2*cos(ni2) - Z1*cos(ni1)) / (Z2-Z1)) / n
    except Exception:
        # Just go with the biggest FFT peak
        i = argmax(F[0:int(N/2)])

    # Convert to Hz
    f0 = i * float(fs) / N

    # Fit it.  We'll catch exceptions here and just returns zeros
    # if something fails with the least squares fit, etc.
    try:
        # first guess for A0, B0 using 3-parameter fit (step c)
        s = zeros(3)
        w = 2*pi*f0

        # Now iterate 7 times (step b, plus 6 iterations of step i)
        for idx in range(7):
            D = c_[cos(w*t), sin(w*t), ones(N),
                   -s[0] * t * sin(w*t) + s[1] * t * cos(w*t) ] # eqn B.16
            s = linalg.lstsq(D, data)[0] # eqn B.18
            w = w + s[3]    # update frequency estimate

        ## Extract results
        A = sqrt(s[0]*s[0] + s[1]*s[1]) # eqn B.21
        f0 = w / (2*pi)
        phi = arctan2(s[0], s[1]) # eqn B.22 (flipped for sin instead of cos)
        C = s[2]
        return (A, f0, phi, C)
    except Exception as e:
        # something broke down; just return zeros
        return (0, 0, 0, 0)

def peak_detect(data, delta = 0.1):
    """Simple min/max peak detection algorithm, taken from my code
    in the disagg.m from the 10-8-5 paper.

    Returns an array of peaks: each peak is a tuple
      (n, p, is_max)
    where n is the row number in 'data', and p is 'data[n]',
    and is_max is True if this is a maximum, False if it's a minimum,
```

```
91      """
92      peaks = [];
93      cur_min = (None, inf)
94      cur_max = (None, -inf)
95      lookformax = False
96      for (n, p) in enumerate(data):
97          if p > cur_max[1]:
98              cur_max = (n, p)
99          if p < cur_min[1]:
100             cur_min = (n, p)
101         if lookformax:
102             if p < (cur_max[1] - delta):
103                 peaks.append((cur_max[0], cur_max[1], True))
104                 cur_min = (n, p)
105                 lookformax = False
106         else:
107             if p > (cur_min[1] + delta):
108                 peaks.append((cur_min[0], cur_min[1], False))
109                 cur_max = (n, p)
110                 lookformax = True
111     return peaks
```

E.2 Management tools

Listing E-3: `nilmtools/cleanup.py (nilm-cleanup)`: Database space estimation and cleanup tool. This tool reads an external configuration file specifying streams and desired amounts of data to save, and can generate an estimation of the disk usage and remove old data.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/cleanup.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1   #!/usr/bin/python
2
3   from nilmdb.utils.printf import *
4   from nilmdb.utils.time import (parse_time, timestamp_to_human,
5                                  timestamp_to_seconds, seconds_to_timestamp)
6   from nilmdb.utils.diskusage import human_size
7   from nilmdb.utils.interval import Interval
8   import nilmdb.client
9   import nilmdb.client.numpyclient
10  import nilmtools
11  import argparse
12  import ConfigParser
13  import sys
14  import collections
15  import fnmatch
16  import re
17
18  def warn(msg, *args):
19      fprintf(sys.stderr, "warning: " + msg + "\n", *args)
20
21  class TimePeriod(object):
22      _units = { 'h': ('hour',  60*60),
23                 'd': ('day',   60*60*24),
24                 'w': ('week',  60*60*24*7),
25                 'm': ('month', 60*60*24*30),
```

```
                'y': ('year',  60*60*24*365) }

    def __init__(self, val):
        for u in self._units:
            if val.endswith(u):
                self.unit = self._units[u][0]
                self.scale = self._units[u][1]
                self.count = float(val[:-len(u)])
                break
        else:
            raise ValueError("unknown units: " + units)

    def seconds(self):
        return self.count * self.scale

    def describe_seconds(self, seconds):
        count = seconds / self.scale
        units = self.unit if count == 1 else (self.unit + "s")
        if count == int(count):
            return sprintf("%d %s", count, units)
        else:
            return sprintf("%.2f %s", count, units)

    def __str__(self):
        return self.describe_seconds(self.seconds())

class StreamCleanupConfig(object):
    def __init__(self, info):
        self.path = info[0]
        self.layout = info[1]
        if info[4] != 0 and info[5] != 0:
            self.rate = info[4] / timestamp_to_seconds(info[5])
        else:
            self.rate = None
        self.keep = None
        self.clean_decimated = True
        self.decimated_from = None
        self.also_clean_paths = []

def main(argv = None):
    parser = argparse.ArgumentParser(
        formatter_class = argparse.RawDescriptionHelpFormatter,
        version = nilmtools.__version__,
        description = """\
    Clean up old data from streams using a configuration file to specify
    which data to remove.

    The format of the config file is as follows:

      [/stream/path]
      keep = 3w         # keep up to 3 weeks of data
      rate = 8000       # optional, used for the --estimate option
      decimated = false # whether to delete decimated data too (default true)

      [*/prep]
      keep = 3.5m       # or 2520h or 105d or 15w or 0.29y

    The suffix for 'keep' is 'h' for hours, 'd' for days, 'w' for weeks,
    'm' for months, or 'y' for years.

    Streams paths may include wildcards.  If a path is matched by more than
    one config section, data from the last config section counts.

    Decimated streams (paths containing '~decim-') are treated specially:
      - They don't match wildcards
      - When deleting data from a parent stream, data is also deleted
        from its decimated streams, unless decimated=false
```

```
93
94      Rate is optional and is only used for the --estimate option.
95      """)
96      parser.add_argument("-u", "--url", action="store",
97                          default="http://localhost/nilmdb/",
98                          help="NilmDB server URL (default: %(default)s)")
99      parser.add_argument("-y", "--yes", action="store_true",
100                         default = False,
101                         help="Actually remove the data (default: no)")
102     parser.add_argument("-e", "--estimate", action="store_true",
103                         default = False,
104                         help="Estimate how much disk space will be used")
105     parser.add_argument("configfile", type=argparse.FileType('r'),
106                         help="Configuration file")
107     args = parser.parse_args(argv)
108
109     # Parse config file
110     config = ConfigParser.RawConfigParser()
111     config.readfp(args.configfile)
112
113     # List all streams
114     client = nilmdb.client.Client(args.url)
115     streamlist = client.stream_list(extended = True)
116
117     # Create config objects
118     streams = collections.OrderedDict()
119     for s in streamlist:
120         streams[s[0]] = StreamCleanupConfig(s)
121         m = re.search(r"^(.*)~decim-[0-9]+$", s[0])
122         if m:
123             streams[s[0]].decimated_from = m.group(1)
124
125     # Build up configuration
126     for section in config.sections():
127         matched = False
128         for path in streams.iterkeys():
129             # Decimated streams only allow exact matches
130             if streams[path].decimated_from and path != section:
131                 continue
132             if not fnmatch.fnmatch(path, section):
133                 continue
134             matched = True
135             options = config.options(section)
136
137             # Keep period (days, weeks, months, years)
138             if 'keep' in options:
139                 streams[path].keep = TimePeriod(config.get(section, 'keep'))
140                 options.remove('keep')
141
142             # Rate
143             if 'rate' in options:
144                 streams[path].rate = config.getfloat(section, 'rate')
145                 options.remove('rate')
146
147             # Decimated
148             if 'decimated' in options:
149                 val = config.getboolean(section, 'decimated')
150                 streams[path].clean_decimated = val
151                 options.remove('decimated')
152
153             for leftover in options:
154                 warn("option '%s' for '%s' is unknown", leftover, section)
155
156         if not matched:
157             warn("config for '%s' did not match any existing streams", section)
158
159     # List all decimated streams in the parent stream's info
```

```
    for path in streams.keys():
        src = streams[path].decimated_from
        if src and src in streams:
            if streams[src].clean_decimated:
                streams[src].also_clean_paths.append(path)
                del streams[path]

    # Warn about streams that aren't getting cleaned up
    for path in streams.keys():
        if streams[path].keep is None or streams[path].keep.seconds() < 0:
            warn("no config for existing stream '%s'", path)
            del streams[path]

    if args.estimate:
        # Estimate disk usage
        total = 0
        for path in streams.keys():
            rate = streams[path].rate
            if not rate or rate < 0:
                warn("unable to estimate disk usage for stream '%s' because "
                     "the data rate is unknown", path)
                continue
            printf("%s:\n", path)
            layout = streams[path].layout
            dtype = nilmdb.client.numpyclient.layout_to_dtype(layout)
            per_row = dtype.itemsize
            per_sec = per_row * rate
            printf("%17s: %s per row, %s rows per second\n",
                   "base rate",
                   human_size(per_row),
                   round(rate,1))
            printf("%17s: %s per hour, %s per day\n",
                   "base size",
                   human_size(per_sec * 3600),
                   human_size(per_sec * 3600 * 24))

            # If we'll be cleaning up decimated data, add an
            # estimation for how much room decimated data takes up.
            if streams[path].clean_decimated:
                d_layout = "float32_" + str(3*(int(layout.split('_')[1])))
                d_dtype = nilmdb.client.numpyclient.layout_to_dtype(d_layout)
                # Assume the decimations will be a factor of 4
                # sum_{k=0..inf} (rate / (n^k)) * d_dtype.itemsize
                d_per_row = d_dtype.itemsize
                factor = 4.0
                d_per_sec = d_per_row * (rate / factor) * (1 / (1 - (1/factor)))
                per_sec += d_per_sec
                printf("%17s: %s per hour, %s per day\n",
                       "with decimation",
                       human_size(per_sec * 3600),
                       human_size(per_sec * 3600 * 24))

            keep = per_sec * streams[path].keep.seconds()
            printf("%17s: %s\n\n",
                   "keep " + str(streams[path].keep), human_size(keep))
            total += keep
        printf("Total estimated disk usage for these streams:\n")
        printf("  %s\n", human_size(total))
        raise SystemExit(0)

    # Do the cleanup
    for path in streams:
        printf("%s: keep %s\n", path, streams[path].keep)

        # Figure out the earliest timestamp we should keep.
        intervals = [ Interval(start, end) for (start, end) in
                      reversed(list(client.stream_intervals(path))) ]
```

```
227             total = 0
228             keep = seconds_to_timestamp(streams[path].keep.seconds())
229             for i in intervals:
230                 total += i.end - i.start
231                 if total <= keep:
232                     continue
233                 remove_before = i.start + (total - keep)
234                 break
235             else:
236                 printf("  nothing to do (only %s of data present)\n",
237                        streams[path].keep.describe_seconds(
238                            timestamp_to_seconds(total)))
239                 continue
240             printf("  removing data before %s\n", timestamp_to_human(remove_before))
241             # Clean in reverse order.  Since we only use the primary stream and not
242             # the decimated streams to figure out which data to remove, removing
243             # the primary stream last means that we might recover more nicely if
244             # we are interrupted and restarted.
245             clean_paths = list(reversed(streams[path].also_clean_paths)) + [ path ]
246             for p in clean_paths:
247                 printf("  removing from %s\n", p)
248                 if args.yes:
249                     client.stream_remove(p, None, remove_before)
250 
251         # All done
252         if not args.yes:
253             printf("Note: specify --yes to actually perform removals\n")
254         return
255 
256     if __name__ == "__main__":
257         main()
```

Listing E-4: `nilmtools/copy_one.py (nilm-copy)`: Stream data copying tool. Supports copying data from one stream or NilmDB instance to another using efficient binary data transfers.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/copy_one.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1   #!/usr/bin/python
2   
3   # This is called copy_one instead of copy to avoid name conflicts with
4   # the Python standard library.
5   
6   import nilmtools.filter
7   import nilmdb.client
8   from nilmdb.client.numpyclient import NumpyClient
9   import numpy as np
10  import sys
11  
12  def main(argv = None):
13      f = nilmtools.filter.Filter()
14      parser = f.setup_parser("Copy a stream")
15      parser.add_argument('-n', '--nometa', action='store_true',
16                          help="Don't copy or check metadata")
17  
18      # Parse arguments
19      try:
20          args = f.parse_args(argv)
21      except nilmtools.filter.MissingDestination as e:
22          print "Source is %s (%s)" % (e.src.path, e.src.layout)
23          print "Destination %s doesn't exist" % (e.dest.path)
```

```
24          print "You could make it with a command like:"
25          print "  nilmtool -u %s create %s %s" % (e.dest.url,
26                                                  e.dest.path, e.src.layout)
27          raise SystemExit(1)
28
29      # Copy metadata
30      if not args.nometa:
31          meta = f.client_src.stream_get_metadata(f.src.path)
32          f.check_dest_metadata(meta)
33
34      # Copy all rows of data using the faster Numpy interfaces
35      extractor = NumpyClient(f.src.url).stream_extract_numpy
36      inserter = NumpyClient(f.dest.url).stream_insert_numpy_context
37      for i in f.intervals():
38          print "Processing", i.human_string()
39          with inserter(f.dest.path, i.start, i.end) as insert_ctx:
40              for data in extractor(f.src.path, i.start, i.end):
41                  insert_ctx.insert(data)
42
43  if __name__ == "__main__":
44      main()
```

Listing E-5: `nilmtools/copy_wildcard.py (nilm-copy-wildcard)`: Stream data copying tool for multiple streams. Wrapper around `nilm-copy` that supports wildcards for copying multiple streams.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/copy_wildcard.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1   #!/usr/bin/python
2
3   # Copy streams between NilmDB servers with wildcards
4
5   import nilmtools.filter
6   import nilmtools.copy_one
7   import nilmdb.client
8   import argparse
9   import fnmatch
10
11  def main(argv = None):
12      f = nilmtools.filter.Filter()
13      # Reuse filter's parser, since it handles most options we need.
14      parser = f.setup_parser(description = """\
15      Copy all streams matching the given wildcard from one host to another.
16
17      Example: %(prog)s -u http://host1/nilmdb -U http://host2/nilmdb /sharon/*
18      """, skip_paths = True)
19      parser.add_argument('-n', '--nometa', action='store_true',
20                          help="Don't copy or check metadata")
21      parser.add_argument("path", action="store", nargs="+",
22                          help='Wildcard paths to copy')
23      args = parser.parse_args(argv)
24
25      # Verify arguments
26      if args.dest_url is None:
27          parser.error("must provide both source and destination URL")
28      client_src = nilmdb.client.Client(args.url)
29      client_dest = nilmdb.client.Client(args.dest_url)
30      if client_src.geturl() == client_dest.geturl():
31          parser.error("source and destination URL must be different")
32      print "Source URL:", client_src.geturl()
33      print "  Dest URL:", client_dest.geturl()
```

```
34
35      # Find matching streams
36      matched = []
37      for path in args.path:
38          matched.extend([s for s in client_src.stream_list(extended = True)
39                          if fnmatch.fnmatch(s[0], path)
40                          and s not in matched])
41
42      # Create destination streams if they don't exist
43      for stream in matched:
44          src = nilmtools.filter.StreamInfo(client_src.geturl(), stream)
45          dest = nilmtools.filter.get_stream_info(client_dest, src.path)
46          if not dest:
47              print "Creating destination stream", src.path
48              client_dest.stream_create(src.path, src.layout)
49
50      # Copy them all by running the "copy" tool as if it were
51      # invoked from the command line.
52      for stream in matched:
53          new_argv = ["--url", client_src.geturl(),
54                      "--dest-url", client_dest.geturl() ]
55          if args.start:
56              new_argv.extend(["--start", "@" + repr(args.start)])
57          if args.end:
58              new_argv.extend(["--end", "@" + repr(args.end)])
59          if args.dry_run:
60              new_argv.extend(["--dry-run"])
61          if args.nometa:
62              new_argv.extend(["--nometa"])
63          if args.force_metadata:
64              new_argv.extend(["--force-metadata"])
65          new_argv.extend([stream[0], stream[0]])
66          try:
67              nilmtools.copy_one.main(new_argv)
68          except SystemExit as e:
69              # Ignore SystemExit which could be raised on --dry-run
70              if e.code != 0:
71                  raise
72
73  if __name__ == "__main__":
74      main()
```

Listing E-6: `nilmtools/pipewatch.py (nilm-pipewatch)`: Helper tool to monitor a pipeline of two commands. Ensures that the pipeline is executed only once concurrently, regardless of how many times the tool is started. Intended for use with `nilm-insert`.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/pipewatch.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1   #!/usr/bin/python
2
3   import nilmdb.client
4   from nilmdb.utils.printf import *
5   import nilmdb.utils.lock
6   import nilmtools
7
8   import time
9   import sys
10  import os
11  import argparse
```

```
import subprocess
import tempfile
import threading
import select
import signal
import Queue
import daemon

def parse_args(argv = None):
    parser = argparse.ArgumentParser(
        formatter_class = argparse.ArgumentDefaultsHelpFormatter,
        version = nilmtools.__version__,
        description = """\
    Pipe data from 'generator' to 'consumer'.  This is intended to be
    executed frequently from cron, and will exit if another copy is
    already running.  If 'generator' or 'consumer' returns an error,
    or if 'generator' stops sending data for a while, it will exit.

    Intended for use with ethstream (generator) and nilm-insert
    (consumer).  Commands are executed through the shell.
    """)
    parser.add_argument("-d", "--daemon", action="store_true",
                        help="Run in background")
    parser.add_argument("-l", "--lock", metavar="FILENAME", action="store",
                        default=tempfile.gettempdir() +
                        "/nilm-pipewatch.lock",
                        help="Lock file for detecting running instance")
    parser.add_argument("-t", "--timeout", metavar="SECONDS", action="store",
                        type=float, default=30,
                        help="Restart if no output from " +
                        "generator for this long")
    group = parser.add_argument_group("commands to execute")
    group.add_argument("generator", action="store",
                       help="Data generator (e.g. \"ethstream -r 8000\")")
    group.add_argument("consumer", action="store",
                       help="Data consumer (e.g. \"nilm-insert /foo/bar\")")
    args = parser.parse_args(argv)

    return args

def reader_thread(queue, fd):
    # Read from a file descriptor, write to queue.
    try:
        while True:
            (r, w, x) = select.select([fd], [], [fd], 0.25)
            if x:
                raise Exception # generator died?
            if not r:
                # short timeout -- just try again.  This is to catch the
                # fd being closed elsewhere, which is only detected
                # when select restarts.
                continue
            data = os.read(fd, 65536)
            if data == "": # generator EOF
                raise Exception
            queue.put(data)
    except Exception:
        queue.put(None)

def watcher_thread(queue, procs):
    # Put None in the queue if either process dies
    while True:
        for p in procs:
            if p.poll() is not None:
                queue.put(None)
                return
        time.sleep(0.25)
```

```
def pipewatch(args):
    # Run the processes, etc
    with open(os.devnull, "r") as devnull:
        generator = subprocess.Popen(args.generator, shell = True,
                                     bufsize = -1, close_fds = True,
                                     stdin = devnull,
                                     stdout = subprocess.PIPE,
                                     stderr = None,
                                     preexec_fn = os.setpgrp)
        consumer = subprocess.Popen(args.consumer, shell = True,
                                    bufsize = -11, close_fds = True,
                                    stdin = subprocess.PIPE,
                                    stdout = None,
                                    stderr = None,
                                    preexec_fn = os.setpgrp)

        queue = Queue.Queue(maxsize = 4)
        reader = threading.Thread(target = reader_thread,
                                  args = (queue, generator.stdout.fileno()))
        reader.start()
        watcher = threading.Thread(target = watcher_thread,
                                   args = (queue, [generator, consumer]))
        watcher.start()
        try:
            while True:
                try:
                    data = queue.get(True, args.timeout)
                    if data is None:
                        break
                    consumer.stdin.write(data)
                except Queue.Empty:
                    # Timeout: kill the generator
                    fprintf(sys.stderr, "pipewatch: timeout\n")
                    generator.terminate()
                    break

            generator.stdout.close()
            consumer.stdin.close()
        except IOError:
            fprintf(sys.stderr, "pipewatch: I/O error\n")

        def kill(proc):
            # Wait for a process to end, or kill it
            def poll_timeout(proc, timeout):
                for x in range(1+int(timeout / 0.1)):
                    if proc.poll() is not None:
                        break
                    time.sleep(0.1)
                return proc.poll()
            try:
                if poll_timeout(proc, 0.5) is None:
                    os.killpg(proc.pid, signal.SIGTERM)
                    if poll_timeout(proc, 0.5) is None:
                        os.killpg(proc.pid, signal.SIGKILL)
            except OSError:
                pass
            return poll_timeout(proc, 0.5)

        # Wait for them to die, or kill them
        cret = kill(consumer)
        gret = kill(generator)

        # Consume all remaining data in the queue until the reader
        # and watcher threads are done
        while reader.is_alive() or watcher.is_alive():
            queue.get(True, 0.1)
```

```
146
147         fprintf(sys.stderr, "pipewatch: generator returned %d, " +
148                 "consumer returned %d\n", gret, cret)
149         if gret == 0 and cret == 0:
150             sys.exit(0)
151         sys.exit(1)
152
153 def main(argv = None):
154     args = parse_args(argv)
155
156     lockfile = open(args.lock, "w")
157     if not nilmdb.utils.lock.exclusive_lock(lockfile):
158         printf("pipewatch process already running (according to %s)\n",
159                args.lock)
160         sys.exit(0)
161     try:
162         # Run as a daemon if requested, otherwise run directly.
163         if args.daemon:
164             with daemon.DaemonContext(files_preserve = [ lockfile ]):
165                 pipewatch(args)
166         else:
167             pipewatch(args)
168     finally:
169         # Clean up lockfile
170         try:
171             os.unlink(args.lock)
172         except OSError:
173             pass
174
175 if __name__ == "__main__":
176     main()
```

E.3 Filters

Listing E-7: `nilmtools/decimate.py (nilm-decimate)`: Stream decimation tool. Performs a single level of the decimation operation described in Section 2.4.1.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/decimate.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1  #!/usr/bin/python
2
3  import nilmtools.filter
4  import nilmdb.client
5  import numpy as np
6  import operator
7
8  def main(argv = None):
9      f = nilmtools.filter.Filter()
10     parser = f.setup_parser("Decimate a stream")
11     group = parser.add_argument_group("Decimate options")
12     group.add_argument('-f', '--factor', action='store', default=4, type=int,
13                        help='Decimation factor (default: %(default)s)')
14
15     # Parse arguments
16     try:
17         args = f.parse_args(argv)
18     except nilmtools.filter.MissingDestination as e:
```

```
        # If no destination, suggest how to create it by figuring out
        # a recommended layout.
        src = e.src
        dest = e.dest
        print "Source is %s (%s)" % (src.path, src.layout)
        print "Destination %s doesn't exist" % (dest.path)
        if "decimate_source" in f.client_src.stream_get_metadata(src.path):
            rec = src.layout
        elif 'int32' in src.layout_type or 'float64' in src.layout_type:
            rec = 'float64_' + str(src.layout_count * 3)
        else:
            rec = 'float32_' + str(src.layout_count * 3)
        print "You could make it with a command like:"
        print "  nilmtool -u %s create %s %s" % (e.dest.url,
                                                 e.dest.path, rec)
        raise SystemExit(1)

    if not (args.factor >= 2):
        raise Exception("factor needs to be 2 or more")

    f.check_dest_metadata({ "decimate_source": f.src.path,
                            "decimate_factor": args.factor })

    # If source is decimated, we have to decimate a bit differently
    if "decimate_source" in f.client_src.stream_get_metadata(args.srcpath):
        again = True
    else:
        again = False
    f.process_numpy(decimate, args = (args.factor, again))

def decimate(data, interval, args, insert_function, final):
    """Decimate data"""
    (factor, again) = args
    (n, m) = data.shape

    # Figure out which columns to use as the source for mean, min, and max,
    # depending on whether this is the first decimation or we're decimating
    # again.  Note that we include the timestamp in the means.
    if again:
        c = (m - 1) // 3
        # e.g. c = 3
        # ts mean1 mean2 mean3 min1 min2 min3 max1 max2 max3
        mean_col = slice(0, c + 1)
        min_col = slice(c + 1, 2 * c + 1)
        max_col = slice(2 * c + 1, 3 * c + 1)
    else:
        mean_col = slice(0, m)
        min_col = slice(1, m)
        max_col = slice(1, m)

    # Discard extra rows that aren't a multiple of factor
    n = n // factor * factor
    data = data[:n,:]

    # Reshape it into 3D so we can process 'factor' rows at a time
    data = data.reshape(n // factor, factor, m)

    # Fill the result
    out = np.c_[ np.mean(data[:,:,mean_col], axis=1),
                 np.min(data[:,:,min_col], axis=1),
                 np.max(data[:,:,max_col], axis=1) ]

    insert_function(out)
    return n

if __name__ == "__main__":
    main()
```

Listing E-8: `nilmtools/decimate_auto.py (nilm-decimate-auto)`: Automatic multiple-level stream decimation tool. Performs repeated decimations using `nilm-decimate` until the final decimation level has fewer than 500 data points.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/decimate_auto.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
#!/usr/bin/python

import nilmtools.filter
import nilmtools.decimate
import nilmdb.client
import argparse
import fnmatch

def main(argv = None):
    parser = argparse.ArgumentParser(
        formatter_class = argparse.RawDescriptionHelpFormatter,
        version = nilmtools.__version__,
        description = """\
    Automatically create multiple decimations from a single source
    stream, continuing until the last decimated level contains fewer
    than 500 points total.

    Wildcards and multiple paths are accepted.  Decimated paths are
    ignored when matching wildcards.
    """)
    parser.add_argument("-u", "--url", action="store",
                        default="http://localhost/nilmdb/",
                        help="NilmDB server URL (default: %(default)s)")
    parser.add_argument("-f", "--factor", action="store", default=4, type=int,
                        help='Decimation factor (default: %(default)s)')
    parser.add_argument("-F", "--force-metadata", action="store_true",
                        default = False,
                        help="Force metadata changes if the dest "
                        "doesn't match")
    parser.add_argument("path", action="store", nargs='+',
                        help='Path of base stream')
    args = parser.parse_args(argv)

    # Pull out info about the base stream
    client = nilmdb.client.Client(args.url)

    # Find list of paths to process
    streams = [ unicode(s[0]) for s in client.stream_list() ]
    streams = [ s for s in streams if "~decim-" not in s ]
    paths = []
    for path in args.path:
        new = fnmatch.filter(streams, unicode(path))
        if not new:
            print "error: no stream matched path:", path
            raise SystemExit(1)
        paths.extend(new)

    for path in paths:
        do_decimation(client, args, path)

def do_decimation(client, args, path):
    print "Decimating", path
    info = nilmtools.filter.get_stream_info(client, path)
    if not info:
        raise Exception("path " + path + " not found")

```

```
57      meta = client.stream_get_metadata(path)
58      if "decimate_source" in meta:
59          print "Stream", path, "was decimated from", meta["decimate_source"]
60          print "You need to pass the base stream instead"
61          raise SystemExit(1)
62
63      # Figure out the type we should use for decimated streams
64      if 'int32' in info.layout_type or 'float64' in info.layout_type:
65          decimated_type = 'float64_' + str(info.layout_count * 3)
66      else:
67          decimated_type = 'float32_' + str(info.layout_count * 3)
68
69      # Now do the decimations until we have few enough points
70      factor = 1
71      while True:
72          print "Level", factor, "decimation has", info.rows, "rows"
73          if info.rows <= 500:
74              break
75          factor *= args.factor
76          new_path = "%s~decim-%d" % (path, factor)
77
78          # Create the stream if needed
79          new_info = nilmtools.filter.get_stream_info(client, new_path)
80          if not new_info:
81              print "Creating stream", new_path
82              client.stream_create(new_path, decimated_type)
83
84          # Run the decimation as if it were run from the commandline
85          new_argv = [ "-u", args.url,
86                       "-f", str(args.factor) ]
87          if args.force_metadata:
88              new_argv.extend([ "--force-metadata" ])
89          new_argv.extend([info.path, new_path])
90          nilmtools.decimate.main(new_argv)
91
92          # Update info using the newly decimated stream
93          info = nilmtools.filter.get_stream_info(client, new_path)
94
95      return
96
97  if __name__ == "__main__":
98      main()
```

Listing E-9: `nilmtools/insert.py (nilm-insert)`: High-level live and external data insertion tool. This tool helps with inserting data from external sources, such as data acquisition boards or previously recorded data files.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/insert.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
1   #!/usr/bin/python
2
3   import nilmdb.client
4   from nilmdb.utils.printf import *
5   from nilmdb.utils.time import (parse_time, timestamp_to_human,
6                                  timestamp_to_seconds, seconds_to_timestamp,
7                                  rate_to_period, now as time_now)
8
9   import nilmtools
10  import time
```

```
import sys
import re
import argparse
import subprocess
import textwrap

class ParseError(Exception):
    def __init__(self, filename, error):
        msg = filename + ": " + error
        super(ParseError, self).__init__(msg)

def parse_args(argv = None):
    parser = argparse.ArgumentParser(
        formatter_class = argparse.RawDescriptionHelpFormatter,
        version = nilmtools.__version__,
        description = textwrap.dedent("""\
    Insert large amount of data from an external source like ethstream.

    This code tracks two timestamps:

    (1) The 'data' timestamp is the precise timestamp corresponding to
        a particular row of data, and is the timestamp that gets
        inserted into the database.  It increases by 'data_delta' for
        every row of input.

        'data_delta' can come from one of two sources.  If '--delta'
        is specified, it is pulled from the first column of data.  If
        '--rate' is specified, 'data_delta' is set to a fixed value of
        (1 / rate).

    (2) The 'clock' timestamp is the less precise timestamp that gives
        the absolute time.  It can come from two sources.  If '--live'
        is specified, it is pulled directly from the system clock.  If
        '--file' is specified, it is extracted from the input filename
        every time a new file is opened for read, and from comments
        that appear in the file.

    Small discrepencies between 'data' and 'clock' are ignored.  If
    the 'data' timestamp ever differs from the 'clock' timestamp by
    more than 'max_gap' seconds:

    - If 'data' is running behind, there is a gap in the data, so it
      is stepped forward to match 'clock'.

    - If 'data' is running ahead, there is overlap in the data, and an
      error is raised.  If '--skip' is specified, the current file
      is skipped instead of raising an error.
    """))
    parser.add_argument("-u", "--url", action="store",
                        default="http://localhost/nilmdb/",
                        help="NilmDB server URL (default: %(default)s)")
    group = parser.add_argument_group("Misc options")
    group.add_argument("-D", "--dry-run", action="store_true",
                       help="Parse files, but don't insert any data")
    group.add_argument("-s", "--skip", action="store_true",
                       help="Skip files if the data would overlap")
    group.add_argument("-m", "--max-gap", action="store", default=10.0,
                       metavar="SEC", type=float,
                       help="Max discrepency between clock and data "
                       "timestamps (default: %(default)s)")

    group = parser.add_argument_group("Data timestamp delta")
    exc = group.add_mutually_exclusive_group()
    exc.add_argument("-r", "--rate", action="store", default=8000.0,
                     type=float,
                     help="Data_delta is constant 1/RATE "
                     "(default: %(default)s Hz)")
```

```
78      exc.add_argument("-d", "--delta", action="store_true",
79                       help="Data_delta is the first number in each line")
80
81      group = parser.add_argument_group("Clock timestamp source")
82      exc = group.add_mutually_exclusive_group()
83      exc.add_argument("-l", "--live", action="store_true",
84                       help="Use live system time for clock timestamp")
85      exc.add_argument("-f", "--file", action="store_true", default=True,
86                       help="Use filename or comments for clock timestamp")
87      group.add_argument("-o", "--offset-filename", metavar="SEC",
88                         action="store", default=-3600.0, type=float,
89                         help="Offset to add to filename timestamps "
90                         "(default: %(default)s)")
91      group.add_argument("-O", "--offset-comment", metavar="SEC",
92                         action="store", default=0.0, type=float,
93                         help="Offset to add to comment timestamps "
94                         "(default: %(default)s)")
95
96      group = parser.add_argument_group("Database path")
97      group.add_argument("path", action="store",
98                         help="Path of stream, e.g. /foo/bar")
99
100     group = parser.add_argument_group("Input files")
101     group.add_argument("infile", type=argparse.FileType('r'), nargs='*',
102                        default=[sys.stdin],
103                        help="Input files (default: stdin)")
104
105     args = parser.parse_args(argv)
106
107     printf("     Stream path: %s\n", args.path)
108
109     printf("  Data timestamp: ")
110     if args.delta:
111         printf("delta on each input line\n")
112     else:
113         printf("fixed rate %s Hz\n", repr(args.rate))
114
115     printf(" Clock timestamp: ")
116     if args.live:
117         printf("live system clock\n")
118     else:
119         printf("from filenames and comments\n")
120         printf(" Filename offset: %s seconds\n", repr(args.offset_filename))
121         printf("  Comment offset: %s seconds\n", repr(args.offset_comment))
122
123     printf("         Max gap: %s seconds\n", repr(args.max_gap))
124     if args.dry_run:
125         printf("Dry run (no data will be inserted)\n")
126
127     return args
128
129 def main(argv = None):
130     args = parse_args(argv)
131
132     client = nilmdb.client.Client(args.url)
133
134     # data_ts is the timestamp that we'll use for the current line
135     data_ts_base = 0
136     data_ts_inc = 0
137     data_ts_rate = args.rate
138     data_ts_delta = 0
139     def get_data_ts():
140         if args.delta:
141             return data_ts_base + data_ts_delta
142         else:
143             return data_ts_base + rate_to_period(data_ts_rate,
144                                                  data_ts_inc)
```

```
145
146     # clock_ts is the imprecise "real" timestamp (from the filename,
147     # comments, or system clock)
148     clock_ts = None
149
150     def print_clock_updated():
151         printf("Clock timestamp updated to %s\n", timestamp_to_human(clock_ts))
152         if data_ts_base != 0:
153             diff = get_data_ts() - clock_ts
154             if diff >= 0:
155                 printf("  (data timestamp ahead by %.6f sec)\n",
156                        timestamp_to_seconds(diff))
157             else:
158                 printf("  (data timestamp behind by %.6f sec)\n",
159                        timestamp_to_seconds(-diff))
160
161     offset_filename = seconds_to_timestamp(args.offset_filename)
162     offset_comment = seconds_to_timestamp(args.offset_comment)
163     max_gap = seconds_to_timestamp(args.max_gap)
164
165     with client.stream_insert_context(args.path) as stream:
166         for f in args.infile:
167             filename = f.name
168             printf("Processing %s\n", filename)
169
170             # If the filename ends in .gz, re-open it with gzip to
171             # decompress.
172             if filename.endswith(".gz"):
173                 p = subprocess.Popen(["gzip", "-dc"],
174                                      stdin = f, stdout = subprocess.PIPE)
175                 f = p.stdout
176
177             # Try to get a real timestamp from the filename
178             try:
179                 # Subtract 1 hour because files are created at the end
180                 # of the hour.  Hopefully, we'll be able to use
181                 # internal comments and this value won't matter anyway.
182                 clock_ts = parse_time(filename) + offset_filename
183                 print_clock_updated()
184             except ValueError:
185                 pass
186
187             truncated_lines = 0
188
189             # Read each line
190             for line in f:
191                 # Once in a while a line might be truncated, if we're
192                 # at the end of a file.  Ignore it, but if we ignore
193                 # too many, bail out.
194                 if line[-1] != '\n':
195                     truncated_lines += 1
196                     if truncated_lines > 3:
197                         raise ParseError(filename, "too many short lines")
198                     printf("Ignoring short line in %s\n", filename)
199                     continue
200
201                 # If no content other than the newline, skip it
202                 if len(line) <= 1:
203                     continue
204
205                 # If line starts with a comment, look for a timestamp
206                 if line[0] == '#':
207                     try:
208                         clock_ts = parse_time(line[1:]) + offset_comment
209                         print_clock_updated()
210                     except ValueError:
211                         pass
```

```
212                     continue
213
214                 # If --delta mode, increment data_ts_delta by the
215                 # delta from the file.
216                 if args.delta:
217                     try:
218                         (delta, line) = line.split(None, 1)
219                         data_ts_delta += float(delta)
220                     except ValueError:
221                         raise ParseError(filename, "can't parse delta")
222
223                 # Calculate data_ts for this row
224                 data_ts = get_data_ts()
225
226                 # If inserting live, use clock timestamp
227                 if args.live:
228                     clock_ts = time_now()
229
230                 # If we have a real timestamp, compare it to the data
231                 # timestamp, and make sure things match up.
232                 if clock_ts is not None:
233                     if (data_ts - max_gap) > clock_ts:
234                         # Accumulated line timestamps are in the future.
235                         # If we were to set data_ts=clock_ts, we'd create
236                         # an overlap, so we have to just bail out here.
237                         err = sprintf("Data is coming in too fast: data time "
238                                       "is %s but clock time is only %s",
239                                       timestamp_to_human(data_ts),
240                                       timestamp_to_human(clock_ts))
241                         if args.skip:
242                             printf("%s\n", err)
243                             printf("Skipping the remainder of this file\n")
244                             break
245                         raise ParseError(filename, err)
246
247                     if (data_ts + max_gap) < clock_ts:
248                         # Accumulated line timetamps are in the past.  We
249                         # can just skip some time and leave a gap in the
250                         # data.
251                         if data_ts_base != 0:
252                             printf("Skipping data timestamp forward from "
253                                    "%s to %s to match clock time\n",
254                                    timestamp_to_human(data_ts),
255                                    timestamp_to_human(clock_ts))
256                         stream.finalize()
257                         data_ts_base = data_ts = clock_ts
258                         data_ts_inc = data_ts_delta = 0
259
260                     # Don't use this clock time anymore until we update it
261                     clock_ts = None
262
263                 if data_ts_base == 0:
264                     raise ParseError(filename, "No idea what timestamp to use")
265
266                 # This line is legit, so increment timestamp (for --rate)
267                 data_ts_inc += 1
268
269                 # Insert it
270                 if not args.dry_run:
271                     stream.insert("%d %s" % (data_ts, line))
272     print "Done"
273
274 if __name__ == "__main__":
275     main()
```

Listing E-10: `nilmtools/median.py (nilm-median)`: Median filter example tool. Applies a simple median filter to all columns of the input data.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/median.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
#!/usr/bin/python
import nilmtools.filter, scipy.signal

def main(argv = None):
    f = nilmtools.filter.Filter()
    parser = f.setup_parser("Median Filter")
    group = parser.add_argument_group("Median filter options")
    group.add_argument("-z", "--size", action="store", type=int, default=25,
                       help = "median filter size (default %(default)s)")
    group.add_argument("-d", "--difference", action="store_true",
                       help = "store difference rather than filtered values")

    try:
        args = f.parse_args(argv)
    except nilmtools.filter.MissingDestination as e:
        print "Source is %s (%s)" % (e.src.path, e.src.layout)
        print "Destination %s doesn't exist" % (e.dest.path)
        print "You could make it with a command like:"
        print "  nilmtool -u %s create %s %s" % (e.dest.url,
                                                 e.dest.path, e.src.layout)
        raise SystemExit(1)

    meta = f.client_src.stream_get_metadata(f.src.path)
    f.check_dest_metadata({ "median_filter_source": f.src.path,
                            "median_filter_size": args.size,
                            "median_filter_difference": repr(args.difference) })

    f.process_numpy(median_filter, args = (args.size, args.difference))

def median_filter(data, interval, args, insert, final):
    (size, diff) = args
    (rows, cols) = data.shape
    for i in range(cols - 1):
        filtered = scipy.signal.medfilt(data[:, i+1], size)
        if diff:
            data[:, i+1] -= filtered
        else:
            data[:, i+1] = filtered
    insert(data)
    return rows

if __name__ == "__main__":
    main()
```

Listing E-11: `nilmtools/trainola.py (nilm-trainola)`:
"Trainola" exemplar matching tool. Given a parameter block from the NILM Manager, extracts exemplars from specified streams and locates instances of those exemplars in the target data, using a cross-correlation approach.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/trainola.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
#!/usr/bin/python

from nilmdb.utils.printf import *
import nilmdb.client
import nilmtools.filter
import nilmtools.math
from nilmdb.utils.time import (timestamp_to_human,
                               timestamp_to_seconds,
                               seconds_to_timestamp)
from nilmdb.utils import datetime_tz
from nilmdb.utils.interval import Interval

import numpy as np
import scipy
import scipy.signal
from numpy.core.umath_tests import inner1d
import nilmrun
from collections import OrderedDict
import sys
import time
import functools
import collections

class DataError(ValueError):
    pass

def build_column_mapping(colinfo, streaminfo):
    """Given the 'columns' list from the JSON data, verify and
    pull out a dictionary mapping for the column names/numbers."""
    columns = OrderedDict()
    for c in colinfo:
        col_num = c['index'] + 1  # skip timestamp
        if (c['name'] in columns.keys() or col_num in columns.values()):
            raise DataError("duplicated columns")
        if (c['index'] < 0 or c['index'] >= streaminfo.layout_count):
            raise DataError("bad column number")
        columns[c['name']] = col_num
    if not len(columns):
        raise DataError("no columns")
    return columns

class Exemplar(object):
    def __init__(self, exinfo, min_rows = 10, max_rows = 100000):
        """Given a dictionary entry from the 'exemplars' input JSON,
        verify the stream, columns, etc.  Then, fetch all the data
        into self.data."""

        self.name = exinfo['name']
        self.url = exinfo['url']
        self.stream = exinfo['stream']
        self.start = exinfo['start']
        self.end = exinfo['end']
        self.dest_column = exinfo['dest_column']

```

```
        # Get stream info
        self.client = nilmdb.client.numpyclient.NumpyClient(self.url)
        self.info = nilmtools.filter.get_stream_info(self.client, self.stream)
        if not self.info:
            raise DataError(sprintf("exemplar stream '%s' does not exist " +
                                    "on server '%s'", self.stream, self.url))

        # Build up name => index mapping for the columns
        self.columns = build_column_mapping(exinfo['columns'], self.info)

        # Count points
        self.count = self.client.stream_count(self.stream, self.start, self.end)

        # Verify count
        if self.count == 0:
            raise DataError("No data in this exemplar!")
        if self.count < min_rows:
            raise DataError("Too few data points: " + str(self.count))
        if self.count > max_rows:
            raise DataError("Too many data points: " + str(self.count))

        # Extract the data
        datagen = self.client.stream_extract_numpy(self.stream,
                                                   self.start, self.end,
                                                   self.info.layout,
                                                   maxrows = self.count)
        self.data = list(datagen)[0]

        # Extract just the columns that were specified in self.columns,
        # skipping the timestamp.
        extract_columns = [ value for (key, value) in self.columns.items() ]
        self.data = self.data[:,extract_columns]

        # Fix the column indices in e.columns, since we removed/reordered
        # columns in self.data
        for n, k in enumerate(self.columns):
            self.columns[k] = n

        # Subtract the means from each column
        self.data = self.data - self.data.mean(axis=0)

        # Get scale factors for each column by computing dot product
        # of each column with itself.
        self.scale = inner1d(self.data.T, self.data.T)

        # Ensure a minimum (nonzero) scale and convert to list
        self.scale = np.maximum(self.scale, [1e-9]).tolist()

    def __str__(self):
        return sprintf("\"%s\" %s [%s] %s rows",
                       self.name, self.stream, ",".join(self.columns.keys()),
                       self.count)

def timestamp_to_short_human(timestamp):
    dt = datetime_tz.datetime_tz.fromtimestamp(timestamp_to_seconds(timestamp))
    return dt.strftime("%H:%M:%S")

def trainola_matcher(data, interval, args, insert_func, final_chunk):
    """Perform cross-correlation match"""
    ( src_columns, dest_count, exemplars ) = args
    nrows = data.shape[0]

    # We want at least 10% more points than the widest exemplar.
    widest = max([ x.count for x in exemplars ])
    if (widest * 1.1) > nrows:
        return 0

```

```
    # This is how many points we'll consider valid in the
    # cross-correlation.
    valid = nrows + 1 - widest
    matches = collections.defaultdict(list)

    # Try matching against each of the exemplars
    for e in exemplars:
        corrs = []

        # Compute cross-correlation for each column
        for col_name in e.columns:
            a = data[:, src_columns[col_name]]
            b = e.data[:, e.columns[col_name]]
            corr = scipy.signal.fftconvolve(a, np.flipud(b), 'valid')[0:valid]

            # Scale by the norm of the exemplar
            corr = corr / e.scale[e.columns[col_name]]
            corrs.append(corr)

        # Find the peaks using the column with the largest amplitude
        biggest = e.scale.index(max(e.scale))
        peaks = nilmtools.math.peak_detect(corrs[biggest], 0.1)

        # To try to reduce false positives, discard peaks where
        # there's a higher-magnitude peak (either min or max) within
        # one exemplar width nearby.
        good_peak_locations = []
        for (i, (n, p, is_max)) in enumerate(peaks):
            if not is_max:
                continue
            ok = True
            # check up to 'e.count' rows before this one
            j = i-1
            while ok and j >= 0 and peaks[j][0] > (n - e.count):
                if abs(peaks[j][1]) > abs(p):
                    ok = False
                j -= 1

            # check up to 'e.count' rows after this one
            j = i+1
            while ok and j < len(peaks) and peaks[j][0] < (n + e.count):
                if abs(peaks[j][1]) > abs(p):
                    ok = False
                j += 1

            if ok:
                good_peak_locations.append(n)

        # Now look at all good peaks
        for row in good_peak_locations:
            # Correlation for each column must be close enough to 1.
            for (corr, scale) in zip(corrs, e.scale):
                # The accepted distance from 1 is based on the relative
                # amplitude of the column.  Use a linear mapping:
                # scale 1.0 -> distance 0.1
                # scale 0.0 -> distance 1.0
                distance = 1 - 0.9 * (scale / e.scale[biggest])
                if abs(corr[row] - 1) > distance:
                    # No match
                    break
            else:
                # Successful match
                matches[row].append(e)

    # Insert matches into destination stream.
    matched_rows = sorted(matches.keys())
    out = np.zeros((len(matched_rows), dest_count + 1))
```

```
189
190     for n, row in enumerate(matched_rows):
191         # Fill timestamp
192         out[n][0] = data[row, 0]
193
194         # Mark matched exemplars
195         for exemplar in matches[row]:
196             out[n, exemplar.dest_column + 1] = 1.0
197
198     # Insert it
199     insert_func(out)
200
201     # Return how many rows we processed
202     valid = max(valid, 0)
203     printf("  [%s] matched %d exemplars in %d rows\n",
204            timestamp_to_short_human(data[0][0]), np.sum(out[:,1:]), valid)
205     return valid
206
207 def trainola(conf):
208     print "Trainola", nilmtools.__version__
209
210     # Load main stream data
211     url = conf['url']
212     src_path = conf['stream']
213     dest_path = conf['dest_stream']
214     start = conf['start']
215     end = conf['end']
216
217     # Get info for the src and dest streams
218     src_client = nilmdb.client.numpyclient.NumpyClient(url)
219     src = nilmtools.filter.get_stream_info(src_client, src_path)
220     if not src:
221         raise DataError("source path '" + src_path + "' does not exist")
222     src_columns = build_column_mapping(conf['columns'], src)
223
224     dest_client = nilmdb.client.numpyclient.NumpyClient(url)
225     dest = nilmtools.filter.get_stream_info(dest_client, dest_path)
226     if not dest:
227         raise DataError("destination path '" + dest_path + "' does not exist")
228
229     printf("Source:\n")
230     printf("  %s [%s]\n", src.path, ",".join(src_columns.keys()))
231     printf("Destination:\n")
232     printf("  %s (%s columns)\n", dest.path, dest.layout_count)
233
234     # Pull in the exemplar data
235     exemplars = []
236     for n, exinfo in enumerate(conf['exemplars']):
237         printf("Loading exemplar %d:\n", n)
238         e = Exemplar(exinfo)
239         col = e.dest_column
240         if col < 0 or col >= dest.layout_count:
241             raise DataError(sprintf("bad destination column number %d\n" +
242                                     "dest stream only has 0 through %d",
243                                     col, dest.layout_count - 1))
244         printf("  %s, output column %d\n", str(e), col)
245         exemplars.append(e)
246     if len(exemplars) == 0:
247         raise DataError("missing exemplars")
248
249     # Verify that the exemplar columns are all represented in the main data
250     for n, ex in enumerate(exemplars):
251         for col in ex.columns:
252             if col not in src_columns:
253                 raise DataError(sprintf("Exemplar %d column %s is not "
254                                         "available in source data", n, col))
255
```

```
256     # Figure out which intervals we should process
257     intervals = ( Interval(s, e) for (s, e) in
258                   src_client.stream_intervals(src_path,
259                                               diffpath = dest_path,
260                                               start = start, end = end) )
261     intervals = nilmdb.utils.interval.optimize(intervals)
262
263     # Do the processing
264     rows = 100000
265     extractor = functools.partial(src_client.stream_extract_numpy,
266                                   src.path, layout = src.layout, maxrows = rows)
267     inserter = functools.partial(dest_client.stream_insert_numpy_context,
268                                  dest.path)
269     start = time.time()
270     processed_time = 0
271     printf("Processing intervals:\n")
272     for interval in intervals:
273         printf("%s\n", interval.human_string())
274         nilmtools.filter.process_numpy_interval(
275             interval, extractor, inserter, rows * 3,
276             trainola_matcher, (src_columns, dest.layout_count, exemplars))
277         processed_time += (timestamp_to_seconds(interval.end) -
278                            timestamp_to_seconds(interval.start))
279     elapsed = max(time.time() - start, 1e-3)
280
281     printf("Done. Processed %.2f seconds per second.\n",
282            processed_time / elapsed)
283
284 def main(argv = None):
285     import simplejson as json
286     import sys
287
288     if argv is None:
289         argv = sys.argv[1:]
290     if len(argv) != 1 or argv[0] == '-h' or argv[0] == '--help':
291         printf("usage: %s [-h] [-v] <json-config-dictionary>\n\n", sys.argv[0])
292         printf("  Where <json-config-dictionary> is a JSON-encoded " +
293                "dictionary string\n")
294         printf("  with exemplar and stream data.\n\n")
295         printf("  See extras/trainola-test-param*.js in the nilmtools " +
296                "repository\n")
297         printf("  for examples.\n")
298         if len(argv) != 1:
299             raise SystemExit(1)
300         raise SystemExit(0)
301
302     if argv[0] == '-v' or argv[0] == '--version':
303         printf("%s\n", nilmtools.__version__)
304         raise SystemExit(0)
305
306     try:
307         # Passed in a JSON string (e.g. on the command line)
308         conf = json.loads(argv[0])
309     except TypeError as e:
310         # Passed in the config dictionary (e.g. from NilmRun)
311         conf = argv[0]
312
313     return trainola(conf)
314
315 if __name__ == "__main__":
316     main()
```

E.4 Additional Filters

The source code for `nilm-prep` and `nilm-sinefit`, part of the `nilmtools` package, can be found in Appendix F.

Appendix F

Sinefit Spectral Envelope Preprocessor Implementation

F.1 Source Code

Listing F-1: `nilm-sinefit`: 4-parameter sine wave fit filter. This code implements a NilmDB filter within the `nilmtools` framework that locates zero crossings using the sine wave fit described in Section 4.3.3.

Git repository: `https://git.jim.sh/jim/lees/nilmtools.git`
Filename: `nilmtools/sinefit.py`
Revision: `f530edd8a08e3f57e40aee7ed5b3719600263af9`

```
#!/usr/bin/python

# Sine wave fitting.
from nilmdb.utils.printf import *
import nilmtools.filter
import nilmtools.math
import nilmdb.client
from nilmdb.utils.time import (timestamp_to_human,
                               timestamp_to_seconds,
                               seconds_to_timestamp)

from numpy import *
from scipy import *
#import pylab as p
import sys

def main(argv = None):
    f = nilmtools.filter.Filter()
    parser = f.setup_parser("Sine wave fitting")
    group = parser.add_argument_group("Sine fit options")
    group.add_argument('-c', '--column', action='store', type=int,
                       help='Column number (first data column is 1)')
```

```
    group.add_argument('-f', '--frequency', action='store', type=float,
                       default=60.0,
                       help='Approximate frequency (default: %(default)s)')
    group.add_argument('-m', '--min-freq', action='store', type=float,
                       help='Minimum valid frequency '
                       '(default: approximate frequency / 2))')
    group.add_argument('-M', '--max-freq', action='store', type=float,
                       help='Maximum valid frequency '
                       '(default: approximate frequency * 2))')
    group.add_argument('-a', '--min-amp', action='store', type=float,
                       default=20.0,
                       help='Minimum signal amplitude (default: %(default)s)')

    # Parse arguments
    try:
        args = f.parse_args(argv)
    except nilmtools.filter.MissingDestination as e:
        rec = "float32_3"
        print "Source is %s (%s)" % (e.src.path, e.src.layout)
        print "Destination %s doesn't exist" % (e.dest.path)
        print "You could make it with a command like:"
        print "  nilmtool -u %s create %s %s" % (e.dest.url, e.dest.path, rec)
        raise SystemExit(1)

    if args.column is None or args.column < 1:
        parser.error("need a column number >= 1")
    if args.frequency < 0.1:
        parser.error("frequency must be >= 0.1")
    if args.min_freq is None:
        args.min_freq = args.frequency / 2
    if args.max_freq is None:
        args.max_freq = args.frequency * 2
    if (args.min_freq > args.max_freq or
        args.min_freq > args.frequency or
        args.max_freq < args.frequency):
        parser.error("invalid min or max frequency")
    if args.min_amp < 0:
        parser.error("min amplitude must be >= 0")

    f.check_dest_metadata({ "sinefit_source": f.src.path,
                            "sinefit_column": args.column })
    f.process_numpy(process, args = (args.column, args.frequency, args.min_amp,
                                     args.min_freq, args.max_freq))

class SuppressibleWarning(object):
    def __init__(self, maxcount = 10, maxsuppress = 100):
        self.maxcount = maxcount
        self.maxsuppress = maxsuppress
        self.count = 0
        self.last_msg = ""

    def _write(self, sec, msg):
        if sec:
            now = timestamp_to_human(seconds_to_timestamp(sec)) + ": "
        else:
            now = ""
        sys.stderr.write(now + msg)

    def warn(self, msg, seconds = None):
        self.count += 1
        if self.count <= self.maxcount:
            self._write(seconds, msg)
        if (self.count - self.maxcount) >= self.maxsuppress:
            self.reset(seconds)

    def reset(self, seconds = None):
        if self.count > self.maxcount:
```

```
            self._write(seconds, sprintf("(%d warnings suppressed)\n",
                                         self.count - self.maxcount))
        self.count = 0

def process(data, interval, args, insert_function, final):
    (column, f_expected, a_min, f_min, f_max) = args
    rows = data.shape[0]

    # Estimate sampling frequency from timestamps
    fs = (rows-1) / (timestamp_to_seconds(data[-1][0]) -
                     timestamp_to_seconds(data[0][0]))

    # Pull out about 3.5 periods of data at once;
    # we'll expect to match 3 zero crossings in each window
    N = max(int(3.5 * fs / f_expected), 10)

    # If we don't have enough data, don't bother processing it
    if rows < N:
        return 0

    warn = SuppressibleWarning(3, 1000)

    # Process overlapping windows
    start = 0
    num_zc = 0
    last_inserted_timestamp = None
    while start < (rows - N):
        this = data[start:start+N, column]
        t_min = timestamp_to_seconds(data[start, 0])
        t_max = timestamp_to_seconds(data[start+N-1, 0])

        # Do 4-parameter sine wave fit
        (A, f0, phi, C) = nilmtools.math.sfit4(this, fs)

        # Check bounds.  If frequency is too crazy, ignore this window
        if f0 < f_min or f0 > f_max:
            warn.warn(sprintf("frequency %s outside valid range %s - %s\n",
                              str(f0), str(f_min), str(f_max)), t_min)
            start += N
            continue

        # If amplitude is too low, results are probably just noise
        if A < a_min:
            warn.warn(sprintf("amplitude %s below minimum threshold %s\n",
                              str(A), str(a_min)), t_min)
            start += N
            continue

        #p.plot(arange(N), this)
        #p.plot(arange(N), A * sin(f0/fs * 2 * pi * arange(N) + phi) + C, 'g')

        # Period starts when the argument of sine is 0 degrees,
        # so we're looking for sample number:
        #     n = (0 - phi) / (f0/fs * 2 * pi)
        zc_n = (0 - phi) / (f0 / fs * 2 * pi)
        period_n = fs/f0

        # Add periods to make N positive
        while zc_n < 0:
            zc_n += period_n

        last_zc = None
        # Mark the zero crossings until we're a half period away
        # from the end of the window
        while zc_n < (N - period_n/2):
            #p.plot(zc_n, C, 'ro')
            t = t_min + zc_n / fs
```

```
157                 if (last_inserted_timestamp is None or
158                     t > last_inserted_timestamp):
159                     insert_function([[seconds_to_timestamp(t), f0, A, C]])
160                     last_inserted_timestamp = t
161                     warn.reset(t)
162                 else:
163                     warn.warn("timestamp overlap\n", t)
164                 num_zc += 1
165                 last_zc = zc_n
166                 zc_n += period_n
167 
168             # Advance the window one quarter period past the last marked
169             # zero crossing, or advance the window by half its size if we
170             # didn't mark any.
171             if last_zc is not None:
172                 advance = min(last_zc + period_n/4, N)
173             else:
174                 advance = N/2
175             #p.plot(advance, C, 'go')
176             #p.show()
177 
178             start = int(round(start + advance))
179 
180         # Return the number of rows we've processed
181         warn.reset(last_inserted_timestamp)
182         if last_inserted_timestamp:
183             now = timestamp_to_human(seconds_to_timestamp(
184                 last_inserted_timestamp)) + ": "
185         else:
186             now = ""
187         printf("%sMarked %d zero-crossings in %d rows\n", now, num_zc, start)
188         return start
189 
190 if __name__ == "__main__":
191     main()
```

Listing F-2: `nilm-prep`: Spectral envelope preprocessor filter. Given raw current waveforms and voltage zero crossing data, this code implements the spectral envelope calculation as described in Section 4.3.2.

Git repository: https://git.jim.sh/jim/lees/nilmtools.git
Filename: nilmtools/prep.py
Revision: f530edd8a08e3f57e40aee7ed5b3719600263af9

```
1  #!/usr/bin/python
2  
3  # Spectral envelope preprocessor.
4  # Requires two streams as input: the original raw data, and sinefit data.
5  
6  from nilmdb.utils.printf import *
7  from nilmdb.utils.time import timestamp_to_human
8  import nilmtools.filter
9  import nilmdb.client
10 from numpy import *
11 import scipy.fftpack
12 import scipy.signal
13 #from matplotlib import pyplot as p
14 import bisect
15 from nilmdb.utils.interval import Interval
16 
17 def main(argv = None):
18     # Set up argument parser
19     f = nilmtools.filter.Filter()
```

```
    parser = f.setup_parser("Spectral Envelope Preprocessor", skip_paths = True)
    group = parser.add_argument_group("Prep options")
    group.add_argument("-c", "--column", action="store", type=int,
                       help="Column number (first data column is 1)")
    group.add_argument("-n", "--nharm", action="store", type=int, default=4,
                       help="number of odd harmonics to compute (default 4)")
    group.add_argument("-N", "--nshift", action="store", type=int, default=1,
                       help="number of shifted FFTs per period (default 1)")
    exc = group.add_mutually_exclusive_group()
    exc.add_argument("-r", "--rotate", action="store", type=float,
                     help="rotate FFT output by this many degrees (default 0)")
    exc.add_argument("-R", "--rotate-rad", action="store", type=float,
                     help="rotate FFT output by this many radians (default 0)")

    group.add_argument("srcpath", action="store",
                       help="Path of raw input, e.g. /foo/raw")
    group.add_argument("sinepath", action="store",
                       help="Path of sinefit input, e.g. /foo/sinefit")
    group.add_argument("destpath", action="store",
                       help="Path of prep output, e.g. /foo/prep")

    # Parse arguments
    try:
        args = f.parse_args(argv)
    except nilmtools.filter.MissingDestination as e:
        rec = "float32_%d" % (e.parsed_args.nharm * 2)
        print "Source is %s (%s)" % (e.src.path, e.src.layout)
        print "Destination %s doesn't exist" % (e.dest.path)
        print "You could make it with a command like:"
        print "  nilmtool -u %s create %s %s" % (e.dest.url, e.dest.path, rec)
        raise SystemExit(1)

    if f.dest.layout_count != args.nharm * 2:
        print "error: need", args.nharm*2, "columns in destination stream"
        raise SystemExit(1)

    # Check arguments
    if args.column is None or args.column < 1:
        parser.error("need a column number >= 1")

    if args.nharm < 1 or args.nharm > 32:
        parser.error("number of odd harmonics must be 1-32")

    if args.nshift < 1:
        parser.error("number of shifted FFTs must be >= 1")

    if args.rotate is not None:
        rotation = args.rotate * 2.0 * pi / 360.0
    else:
        rotation = args.rotate_rad or 0.0

    # Check the sine fit stream
    client_sinefit = nilmdb.client.Client(args.url)
    sinefit = nilmtools.filter.get_stream_info(client_sinefit, args.sinepath)
    if not sinefit:
        raise Exception("sinefit data not found")
    if sinefit.layout != "float32_3":
        raise Exception("sinefit data type is " + sinefit.layout
                        + "; expected float32_3")

    # Check and set metadata in prep stream
    f.check_dest_metadata({ "prep_raw_source": f.src.path,
                            "prep_sinefit_source": sinefit.path,
                            "prep_column": args.column,
                            "prep_rotation": repr(rotation),
                            "prep_nshift": args.nshift })

```

```
    # Find the intersection of the usual set of intervals we'd filter,
    # and the intervals actually present in sinefit data.  This is
    # what we will process.
    filter_int = f.intervals()
    sinefit_int = ( Interval(start, end) for (start, end) in
                    client_sinefit.stream_intervals(
                        args.sinepath, start = f.start, end = f.end) )
    intervals = nilmdb.utils.interval.intersection(filter_int, sinefit_int)

    # Run the process (using the helper in the filter module)
    f.process_numpy(process, args = (client_sinefit, sinefit.path, args.column,
                                     args.nharm, rotation, args.nshift),
                    intervals = intervals)


def process(data, interval, args, insert_function, final):
    (client, sinefit_path, column, nharm, rotation, nshift) = args
    rows = data.shape[0]
    data_timestamps = data[:,0]

    if rows < 2:
        return 0

    last_inserted = [nilmdb.utils.time.min_timestamp]
    def insert_if_nonoverlapping(data):
        """Call insert_function to insert data, but only if this
        data doesn't overlap with other data that we inserted."""
        if data[0][0] <= last_inserted[0]:
            return
        last_inserted[0] = data[-1][0]
        insert_function(data)

    processed = 0
    out = zeros((1, nharm * 2 + 1))
    # Pull out sinefit data for the entire time range of this block
    for sinefit_line in client.stream_extract(sinefit_path,
                                              data[0, 0], data[rows-1, 0]):
        def prep_period(t_min, t_max, rot):
            """
            Compute prep coefficients from time t_min to t_max, which
            are the timestamps of the start and end of one period.
            Results are rotated by an additional extra_rot before
            being inserted into the database.  Returns the maximum
            index processed, or None if the period couldn't be
            processed.
            """
            # Find the indices of data that correspond to (t_min, t_max)
            idx_min = bisect.bisect_left(data_timestamps, t_min)
            idx_max = bisect.bisect_left(data_timestamps, t_max)
            if idx_min >= idx_max or idx_max >= len(data_timestamps):
                return None

            # Perform FFT over those indices
            N = idx_max - idx_min
            d = data[idx_min:idx_max, column]
            F = scipy.fftpack.fft(d) * 2.0 / N

            # If we wanted more harmonics than the FFT gave us, pad with zeros
            if N < (nharm * 2):
                F = r_[F, zeros(nharm * 2 - N)]

            # Fill output data.
            out[0, 0] = round(t_min)
            for k in range(nharm):
                Fk = F[2 * k + 1] * e**(rot * 1j * (k+1))
                out[0, 2 * k + 1] = -imag(Fk) # Pk
                out[0, 2 * k + 2] = real(Fk)  # Qk
```

```
154
155             insert_if_nonoverlapping(out)
156             return idx_max
157
158         # Extract sinefit data to get zero crossing timestamps.
159         # t_min = beginning of period
160         # t_max = end of period
161         (t_min, f0, A, C) = [ float(x) for x in sinefit_line.split() ]
162         t_max = t_min + 1e6 / f0
163
164         # Compute prep over shifted windows of the period
165         # (nshift is typically 1)
166         for n in range(nshift):
167             # Compute timestamps and rotations for shifted window
168             time_shift = n * (t_max - t_min) / nshift
169             shifted_min = t_min + time_shift
170             shifted_max = t_max + time_shift
171             angle_shift = n * 2 * pi / nshift
172             shifted_rot = rotation - angle_shift
173
174             # Run prep computation
175             idx_max = prep_period(shifted_min, shifted_max, shifted_rot)
176             if not idx_max:
177                 break
178             processed = idx_max
179
180     # If we processed no data but there's lots in here, pretend we
181     # processed half of it.
182     if processed == 0 and rows > 10000:
183         processed = rows / 2
184         printf("%s: warning: no periods found; skipping %d rows\n",
185                timestamp_to_human(data[0][0]), processed)
186     else:
187         printf("%s: processed %d of %d rows\n",
188                timestamp_to_human(data[0][0]), processed, rows)
189     return processed
190
191 if __name__ == "__main__":
192     main()
```

Appendix G

Zoom NILM Implementation

G.1 PIC Firmware Source Code

This firmware source code is the implementation of the Zoom NILM physically-windowed sensor architecture, as applied to a current measurement. It targets the Microchip dsPIC33FJ256GP710 microcontroller, using the MPLAB C30 compiler.

Listing G-1: `firmware/zoom.c`: Zoom NILM, main entry point.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/zoom.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "adc.h"
#include "adcext.h"
#include "dac.h"
#include "uart.h"
#include "timer.h"
#include "util.h"
#include "led.h"
#include "mode.h"
#include "calibrate.h"

int main(void)
{
        int jumper;

        config_init();
        led_init();
        led_on();
        calibrate_init();

        /* debug output */
        TRISAbits.TRISA9 = 0;

        adcext_init();
```

```
25          dac_init();
26          dac_write(DAC_MID);
27          adc_init();
28
29          /* Detect jumper from B8 to GND */
30          TRISBbits.TRISB8 = 0;
31          LATBbits.LATB8 = 1;
32          nop(); nop(); nop();
33          jumper = (PORTBbits.RB8 == 0);
34          TRISBbits.TRISB8 = 1;
35
36          /* If jumper present, use MODE_1 */
37          if (jumper)
38                  run_debug();
39          else
40                  run_normal();
41
42          for (;;)
43                  continue;
44  }
```

Listing G-2: `firmware/config.h`: Microcontroller configuration and startup code, header file.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/config.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1   #ifndef CONFIG_H
2   #define CONFIG_H
3
4   #include <p33Fxxxx.h>
5
6   typedef signed char int8_t;
7   typedef unsigned char uint8_t;
8   typedef signed int int16_t;
9   typedef unsigned int uint16_t;
10  typedef signed long int int32_t;
11  typedef unsigned long int uint32_t;
12  typedef signed long long int64_t;
13  typedef unsigned long long uint64_t;
14
15  #define FCY 40000000
16
17  /* define as 0/1 to invert I/O polarity for optocouplers
18     define as 1/0 for normal polarity */
19  #define IO_HIGH 0
20  #define IO_LOW 1
21
22  void config_init(void);
23
24  #define nop() __asm("nop")
25
26  #endif
```

Listing G-3: `firmware/config.c`: Microcontroller configuration and startup code, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/config.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"

/* Configuration words */
_FOSC(FCKSM_CSECMD & POSCMD_XT);
_FOSCSEL(FNOSC_FRC);
_FWDT(FWDTEN_OFF);

void config_init(void)
{
        /* Disable analog inputs */
        AD1PCFGL = 0xffff;
        AD1PCFGH = 0xffff;
        AD2PCFGL = 0xffff;

        /* Configure PLL to multiply from 8 -> 40 MHz */
        PLLFBD = 38;
        CLKDIVbits.PLLPRE = 0;
        CLKDIVbits.PLLPOST = 0;

        /* Switch to XTPLL clock */
        __builtin_write_OSCCONH(0x03);
        __builtin_write_OSCCONL(0x01);

        /* Wait for lock */
        while (!OSCCONbits.LOCK)
                continue;
}
```

Listing G-4: `firmware/calibrate.h`: Calibration routines, header file.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/calibrate.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#ifndef CALIBRATE_H
#define CALIBRATE_H

#define ADC_CLAMP_MIN 256
#define ADC_CLAMP_MAX 1792

#define OVERSAMPLE_COUNT 256

#define DAC_MIN DAC_LOW
#define DAC_MAX DAC_HIGH

#define SEEK_MAX_STEPS 1000
#define SEEK_FUZZ_ADC 100
#define SEEK_FUZZ_DAC (1+((DAC_RANGE * 5) / 65536))

#define CALIBRATE_ADC_ZERO 1024
#define CALIBRATE_ADC_LOW 512
#define CALIBRATE_ADC_HIGH 1536

/* Initialize.  Assume some relatively-safe scaling if no calibration is run */
void calibrate_init(void);

```

```
/* Given the current DAC and ADC values d1 and a1,
   compute a new DAC value d2 to give the desired ADC value a2 */
uint16_t adc_to_dac(uint16_t d1, int16_t a1, int16_t a2, float scale);

/* Calculate a new scale factor given two DAC and ADC points */
float calculate_scale(uint16_t d1, float a1, uint16_t d2, float a2);

/* Seek with the DAC to reach a specific ADC value,
   using current scaling as a starting point. */
uint16_t seek(uint16_t starting_dac, int16_t desired_adc);

/* Perform calibration */
uint16_t do_calibrate(void);

/* Oversample to get a nice ADC value */
float oversample(uint16_t dac);

extern float g_scale;  /* delta(DAC) / delta(ADC) */

#endif
```

Listing G-5: `firmware/calibrate.c`: Calibration routines, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/calibrate.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "calibrate.h"
#include "util.h"
#include "adc.h"
#include "dac.h"
#include "led.h"
#include "timer.h"
#include "uart.h"

//#define DEBUG_CALIBRATION

float g_scale;  /* delta(DAC) / delta(ADC) */

/* Initialize.  Assume some relatively-safe scaling if no calibration is run */
void calibrate_init(void)
{
        g_scale = (DAC_RANGE / 65536.0);  /* 1 count at 16-bit DAC
                                           * means 1 counts at ADC */
}

/* Given the current DAC and ADC values d1 and a1,
   compute a new DAC value d2 to give the desired ADC value a2 */
uint16_t adc_to_dac(uint16_t d1, int16_t a1, int16_t a2, float scale)
{
        int32_t delta;
        int32_t d2;

        delta = (int32_t)((a2 - a1) * scale + 0.5);
        d2 = d1 + delta;
        return clamp(DAC_MIN, d2, DAC_MAX);
}

/* Calculate a new scale factor given two DAC and ADC points */
float calculate_scale(uint16_t d1, float a1, uint16_t d2, float a2)
{
        float scale;
        float a = a2 - a1;
```

```
38
39          /* Correct for known errors */
40          d1 = dac_get_actual_float(d1);
41          d2 = dac_get_actual_float(d2);
42
43          if (a < 0.1 && a > -0.1)
44                  scale = 1.0;
45          else {
46                  scale = ((float)d2 - d1) / a;
47                  if (scale < 0.01)
48                          scale = 0.01;
49                  if (scale > 20)
50                          scale = 20;
51          }
52          return scale;
53  }
54
55  /* Seek with the DAC to reach a specific ADC value.  Uses g_scale as
56     an initial guess for scaling factor, but adjusts it dynamically. */
57  uint16_t seek(uint16_t starting_dac, int16_t desired_adc)
58  {
59          uint16_t old_dac, dac;
60          int16_t old_adc, adc;
61          float scale = g_scale;
62          int steps = 0;
63
64          dac = starting_dac;
65
66          /* goto current location */
67          dac_write(dac);
68          msleep(1);
69          adc = adc_get();
70
71          while (1)
72          {
73                  /* give up if we're not making progress */
74                  if (steps++ > SEEK_MAX_STEPS) {
75                          // 2 flashes, delay, repeat
76                          led_pattern(0b00101000);
77                          break;
78                  }
79
80                  old_dac = dac;
81                  old_adc = adc;
82
83                  /* jump to the desired value */
84                  dac = adc_to_dac(old_dac, old_adc, desired_adc, scale);
85
86                  /* write it out */
87                  dac_write(dac);
88                  msleep(1);
89                  adc = adc_get();
90
91  #ifdef DEBUG_CALIBRATION
92                  uart1_put_hex16(dac);
93                  uart1_put(' ');
94                  uart1_put_hex16(adc);
95                  uart1_put(' ');
96                  uart1_put_hex16(desired_adc);
97                  uart1_put(' ');
98                  uart1_put_hex32(*(uint32_t *)&scale);
99                  uart1_crlf();
100 #endif
101
102                 /* if we're close, accept it */
103                 if (abs(adc - desired_adc) <= SEEK_FUZZ_ADC) {
104                         led_pattern(0b11111110);
```

```
                        break;
                }

                /* otherwise, if we were within ADC clamp limits, and
                   the DAC changed a non-trivial amount, readjust
                   scale factor */
                if (adc > ADC_CLAMP_MIN && old_adc > ADC_CLAMP_MIN &&
                    adc < ADC_CLAMP_MAX && old_adc < ADC_CLAMP_MAX &&
                    abs((int32_t)dac - old_dac) >= SEEK_FUZZ_DAC) {
                        scale = calculate_scale(old_dac, old_adc, dac, adc);
                }

                /* if we totally overshot the window, cut the scale in half */
                if ((adc < ADC_CLAMP_MIN && old_adc > ADC_CLAMP_MAX) ||
                    (adc > ADC_CLAMP_MAX && old_adc < ADC_CLAMP_MIN))
                {
                        scale *= 0.5;
                }
        }

        return dac;
}

/* Perform calibration */
uint16_t do_calibrate(void)
{
        uint16_t daczero, x1, x2;
        float y1, y2;

        /* Zero ADC */
        daczero = seek((DAC_MIN + DAC_MAX) / 2, CALIBRATE_ADC_ZERO);

        /* Go down take an accurate sample */
        x1 = seek(daczero, CALIBRATE_ADC_LOW);
        y1 = oversample(x1);

        /* Go up and take an accurate sample */
        x2 = seek(daczero, CALIBRATE_ADC_HIGH);
        y2 = oversample(x2);

        /* Calculate scale */
        g_scale = calculate_scale(x1, y1, x2, y2);

#ifdef DEBUG_CALIBRATION
        uart1_put_string("calibrate x1=");
        uart1_put_dec(x1);
        uart1_put_string(" y1=");
        uart1_put_float(y1);
        uart1_put_string(" x2=");
        uart1_put_dec(x2);
        uart1_put_string(" y2=");
        uart1_put_float(y2);
        uart1_put_string(" scale=");
        uart1_put_float(g_scale);
        uart1_crlf();
#endif

        /* Return to zero position */
        dac_write(daczero);

        return daczero;
}

/* Oversample to get a nice ADC value */
float oversample(uint16_t dac)
{
        int i;
```

```
172         int32_t sum = 0;
173
174         for (i = 0; i < OVERSAMPLE_COUNT; i++) {
175                 dac_write(dac);
176                 msleep(1);
177                 sum += adc_get();
178         }
179
180         return (float)sum / (float)OVERSAMPLE_COUNT;
181 }
```

Listing G-6: `firmware/mode.h`: Mode selection support, header file. The main code can run in one of two modes, based on the status of a jumper at startup.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/mode.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1 #ifndef MODE_H
2 #define MODE_H
3
4 #include "config.h"
5
6 void run_normal(void);
7 void run_debug(void);
8
9 #endif
```

Listing G-7: `firmware/mode_debug.c`: Debug mode, implementation. Debug mode provides for testing and control via interactive commands sent over the serial port.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/mode_debug.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  #include "config.h"
2  #include "adc.h"
3  #include "dac.h"
4  #include "uart.h"
5  #include "timer.h"
6  #include <stdio.h>
7  #include <math.h>
8  #include "calibrate.h"
9  #include "util.h"
10 #include "led.h"
11 #include "mode.h"
12
13 static uint16_t dac = DAC_MID;
14
15 void sweep(void)
16 {
17         int32_t d;
18         int16_t a;
19
20 #define SWEEP ((DAC_HIGH - DAC_LOW) * (int32_t)2000 / 65535)
21
22         /* sweep range */
```

```
        for (d = (int32_t)dac - SWEEP; d < (int32_t)dac + SWEEP; d++) {
                if (d < DAC_LOW) {
                        uart1_put_dec(DAC_LOW);
                        uart1_put_string(" 0\r\n");
                        continue;
                }
                if (d > DAC_HIGH) {
                        uart1_put_dec(DAC_HIGH);
                        uart1_put_string(" 0\r\n");
                        continue;
                }
                dac_write(d);
                msleep(1);
                a = adc_get();
                uart1_put_dec(d);
                uart1_put(' ');
                uart1_put_dec(a);
                uart1_crlf();
        }
}

void run_debug(void)
{
        int16_t adc;
        int32_t v;
        char buf[4];
        uart1_init(115200);

        led_pattern(0b10101010);

        uart1_put_string("Zoom NILM Debug\r\n");

        while (1) {
                dac = dac & DAC_HIGH; // mask off invalid bits
                dac_write(dac);
                uart1_put_hex16(dac);
                uart1_put(' ');
                uart1_put_dec(dac_get_actual_16bit(dac));
                uart1_put(' ');
                uart1_put(' ');
                adc = adc_get();
                uart1_put_hex16(adc);
                uart1_put(' ');
                uart1_put_dec(adc);
                uart1_crlf();
                switch (uart1_get()) {

                // small step
                case '[':
                        dac--;
                        break;
                case ']':
                        dac++;
                        break;

                // medium step
                case '-':
                        dac -= 16;
                        break;
                case '+':
                case '=':
                        dac += 16;
                        break;

                // big step
                case ',':
                case '<':
```

```
                dac -= 1024;
                break;
        case '.':
        case '>':
                dac += 1024;
                break;

        // set DAC to midpoint
        case '0':
                dac = DAC_MID;
                break;

        // set DAC to specified hex value
        case 'v':
        case 'V':
                buf[0] = uart1_get();
                buf[1] = uart1_get();
                buf[2] = uart1_get();
                buf[3] = uart1_get();
                v = hex_to_u16(buf);
                if (v < 0)
                        uart1_put_string("bad value\r\n");
                else
                        dac = v;
                break;

        // maintain ADC input at zero
        case 'z':
        case 'Z':
                uart1_put_string("zeroing input...\r\n");
                while (!uart1_can_get())
                        dac = seek(dac, 1024);
                uart1_get();
                break;

        // test seeking
        case '1':
                uart1_put_string("seek 512\r\n");
                dac = seek(dac, 512);
                break;

        case '2':
                uart1_put_string("seek 1536\r\n");
                dac = seek(dac, 1536);
                break;

        // run calibration
        case 'c':
        case 'C':
                uart1_put_string("calibrating...\r\n");
                dac = do_calibrate();
                uart1_put_string("new g_scale ");
                uart1_put_float(g_scale);
                uart1_crlf();
                break;

        // sweep DAC
        case 's':
        case 'S':
                uart1_put_string("sweep around ");
                uart1_put_dec(dac);
                uart1_crlf();
                sweep();
                break;
        // dump raw ADC value
        case 'r':
        case 'R':
```

```
157                         while(!uart1_can_get()) {
158                                 uart1_put_hex16(adc_get_raw());
159                                 uart1_crlf();
160                         }
161                         uart1_get();
162                         break;
163                 }
164         }
165 }
```

Listing G-8: `firmware/mode_normal.c`: Normal mode, implementation. Normal mode performs startup calibration followed by the windowing algorithm described in Section 5.4.3.2

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/mode_normal.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  #include "config.h"
2  #include "adc.h"
3  #include "adcext.h"
4  #include "dac.h"
5  #include "uart.h"
6  #include "timer.h"
7  #include <stdio.h>
8  #include <math.h>
9  #include "calibrate.h"
10 #include "util.h"
11 #include "mode.h"
12 #include "led.h"
13 #include "packet.h"
14
15 int send_data = 0;
16 uint16_t send_adc;
17 uint16_t send_dac;
18 int possibly_clamped = 0;
19
20 uint16_t dac_cmd;
21
22 #define TIMER_RATE 8000   /* how often to read the ADC and update DAC */
23 #define PC_RATE 8000      /* how often to send data to the PC */
24
25 #define DEBUG_ISR_TIME
26
27 #define ADC_WINDOW_MIN 512
28 #define ADC_WINDOW_MIN_STEPTO 1280
29
30 #define ADC_WINDOW_MAX 1536
31 #define ADC_WINDOW_MAX_STEPTO 768
32
33 /* Run mode */
34 void TISR_HANDLER(5)
35 {
36         static int count = 0;
37         int16_t v;
38
39 #ifdef DEBUG_ISR_TIME
40         LATAbits.LATA9 = 1;
41 #endif
42         timer_clear_txif(5);
43
44         /* Get most recent sample from 12-bit ADC. */
45         v = adc_get();
```

```
46
47          if (v < ADC_CLAMP_MIN || v >= ADC_CLAMP_MAX)
48                  possibly_clamped = 1;
49
50          /* Send data to PC */
51          if (++count >= (TIMER_RATE / PC_RATE)) {
52                  count = 0;
53
54                  /* Send most recent sample and old DAC value */
55                  send_adc = v;
56                  send_dac = dac_cmd;
57                  send_data = 1;
58          }
59
60  #define WINDOW
61  #ifdef WINDOW
62          /* If ADC value is outside the window, step DAC */
63          if (v < ADC_WINDOW_MIN)
64                  dac_cmd = adc_to_dac(dac_cmd, v, ADC_WINDOW_MIN_STEPTO,
65                                       g_scale);
66          else if (v > ADC_WINDOW_MAX)
67                  dac_cmd = adc_to_dac(dac_cmd, v, ADC_WINDOW_MAX_STEPTO,
68                                       g_scale);
69  #else
70          dac_cmd = adc_to_dac(dac_cmd, v, 1024, g_scale);
71  #endif
72
73          /* Send it out */
74          dac_write(dac_cmd);
75
76  #ifdef DEBUG_ISR_TIME
77          LATAbits.LATA9 = 0;
78  #endif
79  }
80
81  void run_normal(void)
82  {
83          int i;
84          uart1_init(500000);
85
86          led_pattern(0b00110011);
87
88          /* Keep writing zero to the DAC for about 30 seconds after startup,
89             or until we receive a character on the UART */
90          while(uart1_can_get())
91                  uart1_get();
92          for (i = 0; i < 1500; i++) {
93                  dac_write(DAC_MID);
94                  if (uart1_can_get()) {
95                          uart1_get();
96                          break;
97                  }
98                  msleep(10);
99          }
100
101         led_on();
102
103         /* Assume startup current is 0 */
104         msleep(100);
105         dac_cmd = do_calibrate();
106
107         timer_setup_16bit(5, TIMER_RATE, 1);
108         timer_set_priority(5, 6);
109
110         while(1) {
111                 if (send_data) {
112                         /* There's data to send.  Disable the ISR briefly
```

```
                           while we grab it */
                        uint16_t a, d, o;
                        disable_int({
                                        if (possibly_clamped) {
                                                /* Mark a possible
                                                 * overflow in the
                                                 * output */
                                                o = 1;
                                                possibly_clamped = 0;
                                        } else o = 0;
                                        a = send_adc;
                                        d = send_dac;
                                        send_data = 0;
                                });
                        packet_send_adc_dac(a, d, o);
                }
                if (uart1_can_get()) {
                        switch (uart1_get()) {
                        case 'c':
                                disable_int({
                                                send_data = 0;
                                                dac_cmd = do_calibrate();
                                        });
                                packet_send_calibration(g_scale);
                        default:
                                break;
                        }
                }
        }
}
```

Listing G-9: `firmware/adc.h`: 12-bit ADC driver, header file.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/adc.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#ifndef ADC_H
#define ADC_H

#include "config.h"

/* Initialize external 12-bit ADC (AD7450) */
void adc_init(void);

/* Trigger conversion and return it.
   Result is a signed value (-2048 to +2047) */
int16_t adc_get(void);

/* Trigger conversion and get raw 16-bit value from ADC */
uint16_t adc_get_raw(void);

#endif
```

Listing G-10: `firmware/adc.c`: 12-bit ADC driver, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/adc.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "adc.h"
#include "timer.h"

/* Currently using software SPI because it's easy.
   Consider using hardware SPI with DMA on a timer, though. */

#define TRIS_SDATA TRISFbits.TRISF7
#define R_SDATA PORTFbits.RF7
#define TRIS_SCLK TRISFbits.TRISF6
#define LAT_SCLK LATFbits.LATF6
#define TRIS_CS TRISBbits.TRISB2
#define LAT_CS LATBbits.LATB2

void adc_init(void)
{
        TRIS_SDATA = 1;
        LAT_CS = IO_HIGH;
        TRIS_CS = 0;
        LAT_SCLK = IO_HIGH;
        TRIS_SCLK = 0;
}

int16_t adc_get(void)
{
        uint16_t v = 0;
        int i;

        LAT_CS = IO_LOW;
        for (i = 0; i < 16; i++) {
                v <<= 1;
                if (R_SDATA == IO_HIGH)
                        v |= 1;
                LAT_SCLK = IO_LOW;
                nop(); nop(); nop();
                LAT_SCLK = IO_HIGH;
        }
        LAT_CS = IO_HIGH;

        /* Sign-extend the 12-bit value */
        if (v & 0x0800)
                v |= 0xF000;
        else
                v &= ~0xF000;
        return (int16_t) v;
}

uint16_t adc_get_raw(void)
{
        uint16_t v = 0;
        int i;

        LAT_CS = IO_LOW;
        for (i = 0; i < 16; i++) {
                v <<= 1;
                if (R_SDATA == IO_HIGH)
                        v |= 1;
                LAT_SCLK = IO_LOW;
                nop(); nop(); nop();
                LAT_SCLK = IO_HIGH;
```

```
        }
        LAT_CS = IO_HIGH;

        return (uint16_t) v;
}
```

Listing G-11: `firmware/adcext.h`: 24-bit ADC driver, header file. This slower, high-resolution ADC can be used to assist with calibration.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/adcext.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#ifndef ADCEXT_H
#define ADCEXT_H

/* Initialize ADC */
void adcext_init(void);

/* Start a conversion if it hasn't already been started.
   Wait for conversion to finish.
   Read the result and return the raw 32-bit value. */
uint32_t adcext_read(void);

/* Convert a raw 32-bit value into a signed result.
   The return value range is -(2^31) to (2^31)-1 */
int32_t adcext_convert(uint32_t raw);

/* Start a new conversion.  If a conversion was already started
   but the result was not read, this does nothing. */
void adcext_start_conversion(void);

/* Return 1 if a conversion is in progress, 0 otherwise */
int adcext_is_conversion_ready(void);

#endif
```

Listing G-12: `firmware/adcext.c`: 24-bit ADC driver, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/adcext.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "adcext.h"

/* The ADC (AD7846) is tricky, as new conversions start
   automatically when the previous data read finishes.
   To allow more control over start/read, we stretch the
   read indefinitely by delaying the final SCK edge.
   This means we can't have the hardware do SPI for us. */

/* External serial clock, 2-wire I/O -- tie /EXT low,
   tie ADC SDI for rate selection, tie /CS low, use BUSY for
   interrupt notification if desired */

#define TRIS_SDO TRISBbits.TRISB5
#define R_SDO PORTBbits.RB5
#define TRIS_SCK TRISBbits.TRISB4
#define LAT_SCK LATBbits.LATB4
```

```
#define TRIS_SCS TRISBbits.TRISB3
#define LAT_SCS LATBbits.LATB3

/* Short delays */
#define wait_200ns() do { \
        nop(); nop(); nop(); nop(); nop(); nop(); nop();  nop(); \
        } while(0)
#define wait_25ns() nop()

/* Initialize ADC */
void adcext_init(void)
{
        int32_t i;

        TRIS_SDO = 1;
        LAT_SCK = IO_HIGH;
        TRIS_SCK = 0;
        LAT_SCS = IO_LOW;
        TRIS_SCS = 0;

        /* Startup delay CS down to SCK high t4 = 5000ns */
        for (i = 0; i < 5000 / 25; i++)
                wait_25ns();

        /* We need to monitor BUSY, I think.  With the 2-wire
           interface, we never know if we start in the middle of
       a data output phase. For now, consider reading broken..
           just return early.  XXXX */
        return;

        /* Trigger a dummy read so we're prepared for the
           next conversion */
        (void) adcext_read();
}

/* Start a conversion if it hasn't already been started.
   Wait for conversion to finish.
   Read the result and return the raw 32-bit value. */
uint32_t adcext_read(void)
{
        uint32_t val;
        int i;

        /* Start conversion by completing previous read */
        LAT_SCK = IO_LOW;

        /* Wait tKQMAX for SCK down to SDO valid */
        wait_200ns();

        /* Wait for conversion to finish */
        while (R_SDO == IO_HIGH)
                continue;

        /* Read it out */
        val = 0;
        for (i = 0; i < 32; i++) {
                /* SCK low tLESCK = 25ns */
                wait_25ns();
                LAT_SCK = IO_HIGH;

                /* SCK high tHESCK = 25ns, but
                   we also have SCK down to SDO valid tKQMAX = 200ns?
                   Probably misspecified but wait tKQMAX anyway. */
                wait_200ns();

                val <<= 1;
                if (R_SDO == IO_HIGH)
```

```
85                      val |= 1;
86              /* Leave SCK high on final bit to delay new conversion */
87              if (i < 31)
88                      LAT_SCK = IO_LOW;
89      }
90
91      /* Done */
92      return val;
93  }
94
95  /* Convert a raw 32-bit value into a signed 32-bit result.
96     The return value is full int32 range but only the high
97     24 should be significant: low 3 will always be 0,
98     and the next 5 will be sub-resolution (see datasheet). */
99  int32_t adcext_convert(uint32_t raw)
100 {
101     int sigmsb = (raw >> 28) & 3;
102
103     /* If SIG & MSB, it is a positive overflow */
104     if (sigmsb == 3)
105             return (int32_t)0x7FFFFFFFL;
106     /* If !SIG & !MSB, it is a negative overflow */
107     if (sigmsb == 0)
108             return (int32_t)0x80000000L;
109     /* Shift over EOC,DMY,SIG and return */
110     return ((int32_t)(raw << 3));
111 }
112
113 /* Start a new conversion.  If a conversion was already started
114    but the result was not read, this does nothing. */
115 void adcext_start_conversion(void)
116 {
117     /* If we had a previous conversion ready to read,
118        read it out so we can start a new conversion instead */
119     if (adcext_is_conversion_ready())
120             (void) adcext_read();
121
122     /* Start conversion by completing previous read */
123     LAT_SCK = 0;
124
125     /* Wait tKQMAX for SCK down to SDO valid in case we
126        call adcext_is_conversion_ready right away. */
127     wait_200ns();
128 }
129
130 /* Return 1 if a conversion is finished and ready to be read, 0 otherwise */
131 int adcext_is_conversion_ready(void)
132 {
133     if (LAT_SCK == 0 && R_SDO == 0)
134             return 1;
135     return 0;
136 }
```

Listing G-13: `firmware/dac.h`: 10 or 16-bit DAC driver and conversion routines, header file. This driver supports compile-time selection of DAC type between the 16-bit AD5542, a 10-bit simulation using the AD5542, and a true 10-bit MAX504.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/dac.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```c
#ifndef DAC_H
#define DAC_H

#include "config.h"

#define DAC_TYPE 2

#if DAC_TYPE == 0
  /* AD5542, normal 16-bit range */
  #define DAC_BITS 16
  #define __dac_to_spi_cmd(x) (x)
  #define __dac_to_16bit_equiv(x) (x)
#elif DAC_TYPE == 1
  /* AD5542, fake 10-bit range using random lower bits */
  #define DAC_BITS 10
  #define __dac_to_spi_cmd(x) (dac_lookup[(x)&1023])
  #define __dac_to_16bit_equiv(x) (dac_lookup[(x)&1023])
#elif DAC_TYPE == 2
  /* MAX504, true 10-bit DAC */
  #define DAC_BITS 10
  #define __dac_to_spi_cmd(x) ((x) << 2)
  #define __dac_to_16bit_equiv(x) ((x) << 6)
#else
  #error Unknown DAC type
#endif

#define DAC_LOW 0
#define DAC_HIGH ((uint16_t)(((uint32_t)1 << DAC_BITS) - 1))
#define DAC_MID ((uint16_t)((DAC_LOW + DAC_HIGH + (uint32_t)1) / 2))
#define DAC_RANGE (DAC_HIGH - DAC_LOW + 1)

/* Initialize DAC (AD5542) */
void dac_init(void);

/* Write raw value to DAC:
   DAC_HIGH 4.9998v
   DAC_MID  0v
   DAC_LOW  -5v
 */
void dac_write(uint16_t val);

/* Given a DAC command between DAC_LOW and DAC_HIGH,
   get the actual expected output voltage as a 16-bit value, where:
     0xffff = 4.9998v
     0x8000 = 0v
     0x0000 = -5v
*/
uint16_t dac_get_actual_16bit(uint16_t val);

/* Given a DAC command between DAC_LOW and DAC_HIGH,
   get the actual expected output voltage as a FLOAT, where:
     DAC_HIGH = 4.9998v
     DAC_MID  = 0v
     DAC_LOW  = -5v
   Example: for 10-bit DAC, convert integer command 123 into
   the more accurate 123.45 using known lower bits.
*/
float dac_get_actual_float(uint16_t val);

#endif
```

Listing G-14: `firmware/dac.c`: 10 or 16-bit DAC driver and conversion routines, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/dac.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "dac.h"

/* Initialize DAC (AD5542) */
void dac_init(void)
{
        /* SPI2 */
        IEC2bits.SPI2IE = 0;

        SPI2CON1bits.DISSCK = 0;
        SPI2CON1bits.DISSDO = 0;
        SPI2CON1bits.MODE16 = 1;
        SPI2CON1bits.SMP = 0;
        SPI2CON1bits.CKE = 0;
        SPI2CON1bits.SSEN = 0;
        SPI2CON1bits.CKP = IO_HIGH;
        SPI2CON1bits.MSTEN = 1;
        SPI2CON1bits.SPRE = 4;
        SPI2CON1bits.PPRE = 3;

        /* There's no framed mode that does the normal
       "chip select" behavior, so control /SS2 manually */
        SPI2CON2bits.FRMEN = 0;
        LATGbits.LATG9 = IO_HIGH;
        TRISGbits.TRISG9 = 0;

        SPI2STATbits.SPISIDL = 0;
        SPI2STATbits.SPIEN = 1;

        dac_write(0x0000);
}

static const uint16_t dac_lookup[1024] = {
  #include "lookup.inc"
};

/* Write raw 16-bit desired value to DAC:
   DAC_HIGH 4.9998v
   DAC_MID  0v
   DAC_LOW -5v
 */
void dac_write(uint16_t val)
{
        LATGbits.LATG9 = IO_LOW;
        if (IO_HIGH == 1)
                SPI2BUF = __dac_to_spi_cmd(val);
        else
                SPI2BUF = ~__dac_to_spi_cmd(val);
        while (!SPI2STATbits.SPIRBF)
                continue;
        (void) SPI2BUF;
        LATGbits.LATG9 = IO_HIGH;
}

/* Given a DAC command between DAC_LOW and DAC_HIGH,
   get the actual expected output voltage as a 16-bit value, where:
     0xffff = 4.9998v
     0x8000 = 0v
     0x0000 = -5v
```

```
*/
uint16_t dac_get_actual_16bit(uint16_t val)
{
        return __dac_to_16bit_equiv(val);
}

/* Given a DAC command between DAC_LOW and DAC_HIGH,
   get the actual expected output voltage as a FLOAT, where:
     DAC_HIGH = 4.9998v
     DAC_MID  = 0v
     DAC_LOW  = -5v
   Example: for 10-bit DAC, convert integer command 123 into
   the more accurate 123.45 using known lower bits.
*/
float dac_get_actual_float(uint16_t val)
{
        return __dac_to_16bit_equiv(val) * (DAC_RANGE / 65536.0);
}
```

Listing G-15: `firmware/gen-dac-lookup.pl`: Script to generate DAC lookup table. The DAC code uses the output of this script, `lookup.inc`, to map 10-bit DAC command values to a 16-bit output voltage.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/gen-dac-lookup.pl`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#!/usr/bin/perl

srand(1337);

# constant bits, the rest are filled with random data
$bits = $ARGV[0] || 10;

$rem = 16 - $bits;
for ($i = 0; $i < 2**$bits; $i++)
{
    $out = $i * 2**$rem + int(rand(2**$rem));
    print $out . ",\n";
}
```

Listing G-16: `firmware/led.h`: LED indicator driver, header file. Up to an 8-bit pattern can be blinked out serially on the LED in a repeating fashion, allowing multiple status indicators to be presented to the user.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/led.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#ifndef LED_H
#define LED_H

#include "config.h"

#define LED_BLINK_RATE 8
extern int16_t __led_pattern;

```

```
/* Initialize LED */
void led_init(void);

/* Set a pattern (8-bit binary pattern). */
void led_pattern(uint8_t pattern);

static inline void led_on(void)
{
        __led_pattern = -1;
        PORTBbits.RB13 = 0;
}

static inline void led_off(void)
{
        __led_pattern = -1;
        PORTBbits.RB13 = 1;
}

#endif
```

Listing G-17: `firmware/led.c`: LED indicator driver, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/led.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "led.h"
#include "timer.h"

int16_t __led_pattern;

/* Debug LED */
void TISR_HANDLER(6)
{
        timer_clear_txif(6);

        if (__led_pattern == -1)
                return;

        __led_pattern <<= 1;
        if (__led_pattern & 0x100) {
                PORTBbits.RB13 = 0;  /* on */
                __led_pattern |= 1;
        } else {
                PORTBbits.RB13 = 1;  /* off */
        }

        __led_pattern &= 0xff;
}

void led_init(void)
{
        TRISBbits.TRISB13 = 0;
        PORTBbits.RB13 = 1;
        __led_pattern = -1;

        timer_setup_16bit(6, LED_BLINK_RATE, 1);
        timer_set_priority(6, 1);    /* low priority */
}

/* Set a pattern (8-bit binary pattern). */
void led_pattern(uint8_t pattern)
{
```

```
39          __led_pattern = pattern;
40  }
```

Listing G-18: `firmware/packet.h`: Data serialization for PC communication, header file.

Git repository: https://git.jim.sh/jim/lees/zoom.git
Filename: firmware/packet.h
Revision: e28af0bb6747b0631f28ffd09813c45105e6906c

```
1   #ifndef PACKET_H
2   #define PACKET_H
3
4   #include "config.h"
5
6   #define PACKET_ADC_DAC 0xA0
7   #define PACKET_CALIBRATION 0xA1
8
9   void packet_send_adc_dac(int16_t adc, uint16_t dac, int overflow);
10  void packet_send_calibration(float scale);
11
12  #endif
```

Listing G-19: `firmware/packet.c`: Data serialization for PC communication, implementation.

Git repository: https://git.jim.sh/jim/lees/zoom.git
Filename: firmware/packet.c
Revision: e28af0bb6747b0631f28ffd09813c45105e6906c

```
1   #include "packet.h"
2   #include "uart.h"
3
4   /* 5 byte packets.  5 bytes * 10 bits/byte * 8 khz = 400 kbits/sec */
5
6   void packet_send_adc_dac(int16_t adc, uint16_t dac, int overflow)
7   {
8           char b[4];
9           /* Packet format for DAC/ADC values:
10             10100000  ffffAaaa  aaaaaaaa  Dddddddd  dddddddd
11
12             ffff = flags, default 0000
13             ...1 = possible overflow
14             Aaaaaaaaaaaa = 12-bit ADC value (2s compliment signed)
15             Ddddddddddddddddd = 16-bit DAC command (unsigned)
16          */
17
18          uart1_put(PACKET_ADC_DAC);
19
20          b[0] = (adc & 0x0F00) >> 8;
21          if (overflow)
22                  b[0] |= 0x10;
23          b[1] = adc & 0xFF;
24          b[2] = (dac & 0xFF00) >> 8;
25          b[3] = dac & 0xFF;
26          uart1_put(b[0]);
27          uart1_put(b[1]);
28          uart1_put(b[2]);
29          uart1_put(b[3]);
30  }
```

```
31
32 void packet_send_calibration(float scale)
33 {
34         /* Packet format for calibration data:
35            A1 xx xx xx xx
36
37            where xx xx xx xx = 32-bit floating point value, little
38            endian IEEE 754 */
39
40         uint8_t *b = (uint8_t *)&scale;
41
42         uart1_put(PACKET_CALIBRATION);
43         uart1_put(b[0]);
44         uart1_put(b[1]);
45         uart1_put(b[2]);
46         uart1_put(b[3]);
47 }
```

Listing G-20: `firmware/timer.h`: Timer driver, header file.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/timer.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  #ifndef TIMER_H
2  #define TIMER_H
3
4  #include "config.h"
5
6  /* Setup a 16-bit timer to overflow at the specified frequency
7       timer = 1-9
8       freq = Hz, or 0 to disable the timer.
9       ie = 1 to enable interrupt, 0 to disable
10 */
11 int timer_setup_16bit(int timer, uint32_t freq, int ie);
12
13 #define TISR_HANDLER(x) \
14         __attribute__((__interrupt__,auto_psv)) _T##x##Interrupt(void)
15
16 /* Clear TxIF corresponding to a timer */
17 #define timer_clear_txif(timer) do {                    \
18                 if ((timer) == 1) IFS0bits.T1IF = 0;    \
19                 if ((timer) == 2) IFS0bits.T2IF = 0;    \
20                 if ((timer) == 3) IFS0bits.T3IF = 0;    \
21                 if ((timer) == 4) IFS1bits.T4IF = 0;    \
22                 if ((timer) == 5) IFS1bits.T5IF = 0;    \
23                 if ((timer) == 6) IFS2bits.T6IF = 0;    \
24                 if ((timer) == 7) IFS3bits.T7IF = 0;    \
25                 if ((timer) == 8) IFS3bits.T8IF = 0;    \
26                 if ((timer) == 9) IFS3bits.T9IF = 0;    \
27         } while(0)
28
29
30 /* Set timer interrupt priority, 1-7.  Default is 4, 7 is highest */
31 #define timer_set_priority(timer, pri) do { \
32                 if ((timer) == 1) IPC0bits.T1IP = (pri);        \
33                 if ((timer) == 2) IPC1bits.T2IP = (pri);        \
34                 if ((timer) == 3) IPC2bits.T3IP = (pri);        \
35                 if ((timer) == 4) IPC6bits.T4IP = (pri);        \
36                 if ((timer) == 5) IPC7bits.T5IP = (pri);        \
37                 if ((timer) == 6) IPC11bits.T6IP = (pri);       \
38                 if ((timer) == 7) IPC12bits.T7IP = (pri);       \
39                 if ((timer) == 8) IPC12bits.T8IP = (pri);       \
40                 if ((timer) == 9) IPC13bits.T9IP = (pri);       \
```

```
        } while(0)

/* sleep for between "ms" and "ms+1" milliseconds */
void msleep(int ms);

#endif
```

Listing G-21: `firmware/timer.c`: Timer driver, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/timer.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "timer.h"

/* Setup a 16-bit timer to overflow at the specified frequency
     timer = 1-9
     freq = Hz, or 0 to disable the timer.
     interrupt = 1 to enable interrupt, 0 to disable
*/
int timer_setup_16bit(int timer, uint32_t freq, int ie)
{
        uint32_t period;
        uint16_t prescale;

        if (timer < 1 || timer > 9)
                return -1;

        if (freq == 0) {
                switch(timer) {
                case 1: T1CONbits.TON = 0; return 0;
                case 2: T2CONbits.TON = 0; return 0;
                case 3: T3CONbits.TON = 0; return 0;
                case 4: T4CONbits.TON = 0; return 0;
                case 5: T5CONbits.TON = 0; return 0;
                case 6: T6CONbits.TON = 0; return 0;
                case 7: T7CONbits.TON = 0; return 0;
                case 8: T8CONbits.TON = 0; return 0;
                case 9: T9CONbits.TON = 0; return 0;
                }
        }

        /* Figure out timer prescaler and period values.  Max period
           is 65535 (PRx = 65534) so we can still attain 100% duty
           cycle when using a timer for PWM. */
        if ((period = FCY / (freq * 1L)) <= 65535)
                prescale = 0;
        else if ((period = FCY / (freq * 8L)) <= 65535)
                prescale = 1;
        else if ((period = FCY / (freq * 64L)) <= 65535)
                prescale = 2;
        else if ((period = FCY / (freq * 256L)) <= 65535)
                prescale = 3;
        else
                prescale = 3, period = 65535;
        if (period > 0)
                period -= 1;

        switch (timer) {
#define __timer_setup_case(x) \
case x: \
        T##x##CONbits.TON = 0; \
        T##x##CONbits.TCKPS = prescale; \
```

```
52          PR##x = period; \
53          T##x##CONbits.TON = 1; \
54          break
55                  __timer_setup_case(1);
56                  __timer_setup_case(2);
57                  __timer_setup_case(3);
58                  __timer_setup_case(4);
59                  __timer_setup_case(5);
60                  __timer_setup_case(6);
61                  __timer_setup_case(7);
62                  __timer_setup_case(8);
63                  __timer_setup_case(9);
64  #undef __timer_setup_case
65          }
66
67          /* Enable interrupt if requested */
68          timer_clear_txif(timer);
69          switch (timer) {
70          case 1: IEC0bits.T1IE = ie; break;
71          case 2: IEC0bits.T2IE = ie; break;
72          case 3: IEC0bits.T3IE = ie; break;
73          case 4: IEC1bits.T4IE = ie; break;
74          case 5: IEC1bits.T5IE = ie; break;
75          case 6: IEC2bits.T6IE = ie; break;
76          case 7: IEC3bits.T7IE = ie; break;
77          case 8: IEC3bits.T8IE = ie; break;
78          case 9: IEC3bits.T9IE = ie; break;
79          }
80
81          return 0;
82  }
83
84  /* sleep for between "ms" and "ms+1" milliseconds */
85  void msleep(int ms)
86  {
87          static int initialized = 0;
88
89          if (!initialized) {
90                  timer_setup_16bit(1, 1000, 0);
91                  initialized = 1;
92          }
93
94          /* Very basic, assumes timer1 is set up at 1 khz */
95          while (ms-- >= 0) {
96                  IFS0bits.T1IF = 0;
97                  while(IFS0bits.T1IF == 0)
98                          continue;
99          }
100 }
```

Listing G-22: `firmware/uart.h`: Serial communication (UART) driver, header file.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/uart.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1   #ifndef UART_H
2   #define UART_H
3
4   #include "config.h"
5
6   /* Init */
```

```
void uart_init(int uart, int32_t rate);

/* Blocking sends */
void uart_put(int uart, uint8_t x);
void uart_put_string(int uart, const char *s);
void uart_crlf(int uart);
void uart_put_hex(int uart, uint8_t x);
void uart_put_hex16(int uart, uint16_t x);
void uart_put_hex32(int uart, uint32_t x);
void uart_put_bin(int uart, uint8_t x);
void uart_put_dec(int uart, int32_t x);
void uart_put_float(int uart, float x);

/* Blocking receives */
uint8_t uart_get(int uart);

/* Return true if get/put would not block */
int uart_can_get(int uart);
int uart_can_put(int uart);

/* Helpers to work with a specific uart */
#define uart1_init(x)           uart_init(1, x)
#define uart1_can_get()         uart_can_get(1)
#define uart1_get()             uart_get(1)
#define uart1_put(x)            uart_put(1,x)
#define uart1_put_string(x) uart_put_string(1,x)
#define uart1_crlf()            uart_crlf(1)
#define uart1_put_hex(x)        uart_put_hex(1,x)
#define uart1_put_hex16(x)      uart_put_hex16(1,x)
#define uart1_put_hex32(x)      uart_put_hex32(1,x)
#define uart1_put_bin(x)        uart_put_bin(1,x)
#define uart1_put_dec(x)        uart_put_dec(1,x)
#define uart1_put_float(x)      uart_put_float(1,x)

#define uart2_init(x)           uart_init(2, x)
#define uart2_can_get()         uart_can_get(2)
#define uart2_get()             uart_get(2)
#define uart2_put(x)            uart_put(2,x)
#define uart2_put_string(x) uart_put_string(2,x)
#define uart2_crlf()            uart_crlf(2)
#define uart2_put_hex(x)        uart_put_hex(2,x)
#define uart2_put_hex16(x)      uart_put_hex16(2,x)
#define uart2_put_hex32(x)      uart_put_hex32(2,x)
#define uart2_put_bin(x)        uart_put_bin(2,x)
#define uart2_put_dec(x)        uart_put_dec(2,x)
#define uart2_put_float(x)      uart_put_float(2,x)

#endif
```

Listing G-23: `firmware/uart.c`: Serial communication (UART) driver, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/uart.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "uart.h"
#include "util.h"

void uart_init(int uart, int32_t rate)
{
        int32_t brg;
```

```
8
9           brg = ((FCY + (8 * rate - 1)) / (16 * rate)) - 1;
10          if (brg < 1) brg = 1;
11          if (brg > 65535) brg = 65535;
12
13          if (uart == 1) {
14                  U1MODE = 0;
15                  U1MODEbits.BRGH = 0; // errata: BRGH=1 is broken
16                  U1BRG = brg;
17                  U1STA = 0;
18                  U1MODEbits.UARTEN = 1;
19                  U1STAbits.UTXEN = 1;
20          } else if (uart == 2) {
21                  U2MODE = 0;
22                  U2MODEbits.BRGH = 0;
23                  U2BRG = brg;
24                  U2STA = 0;
25                  U2MODEbits.UARTEN = 1;
26                  U2STAbits.UTXEN = 1;
27          }
28  }
29
30  void uart_put(int uart, uint8_t x)
31  {
32          if (uart == 1) {
33                  while (U1STAbits.UTXBF) continue;
34                  U1TXREG = x;
35          } else if (uart == 2) {
36                  while (U2STAbits.UTXBF) continue;
37                  U2TXREG = x;
38          }
39  }
40
41  int uart_can_get(int uart)
42  {
43          if (uart == 1)
44                  return U1STAbits.URXDA;
45          else if (uart == 2)
46                  return U2STAbits.URXDA;
47          else
48                  return 0;
49  }
50
51  uint8_t uart_get(int uart)
52  {
53          uint8_t data = 0;
54          if (uart == 1) {
55                  while (!U1STAbits.URXDA) continue;
56                  data = U1RXREG;
57                  if (U1STAbits.OERR)
58                          U1STAbits.OERR = 0;
59          } else if (uart == 2) {
60                  while (!U2STAbits.URXDA) continue;
61                  data = U2RXREG;
62                  if (U2STAbits.OERR)
63                          U2STAbits.OERR = 0;
64          }
65          return data;
66  }
67
68  void uart_put_string(int uart, const char *s)
69  {
70          while(s && *s)
71                  uart_put(uart, *s++);
72  }
73
74  void uart_put_bin(int uart, uint8_t x)
```

```
{
        int i;
        for(i=0;i<8;i++) {
                uart_put(uart, (x & 0x80) ? '1' : '0');
                x <<= 1;
        }
}

void uart_put_hex(int uart, uint8_t x)
{
        uart_put(uart, hex[x >> 4]);
        uart_put(uart, hex[x & 15]);
}

void uart_put_hex16(int uart, uint16_t x)
{
        uart_put_hex(uart, (x >> 8) & 0xFF);
        uart_put_hex(uart, (x) & 0xFF);
}

void uart_put_hex32(int uart, uint32_t x)
{
        uart_put_hex(uart, (x >> 24) & 0xFF);
        uart_put_hex(uart, (x >> 16) & 0xFF);
        uart_put_hex(uart, (x >> 8) & 0xFF);
        uart_put_hex(uart, (x) & 0xFF);
}

void uart_put_dec(int uart, int32_t x)
{
        uint32_t val;
        uint32_t place = 1;

        if (x >= 0) {
                val = x;
        } else {
                uart_put(uart, '-');
                val = -x;
        }

        while (val / place > 9)
                place *= 10;
        while (place > 0) {
                uart_put(uart, val / place + '0');
                val %= place;
                place /= 10;
        }
}

void uart_put_float(int uart, float x)
{
        int i;
        int32_t v;

        v = (int32_t) x;
        uart_put_dec(uart, v);
        uart_put(uart, '.');
        for (i = 0; i < 6; i++) {
                x -= v;
                x *= 10;
                v = (int32_t) x;
                uart_put(uart, v + '0');
        }
}

void uart_crlf(int uart)
{
```

```
142         uart_put(uart, '\r');
143         uart_put(uart, '\n');
144 }
```

Listing G-24: `firmware/util.h`: Miscellaneous firmware utilities, header file.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/util.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  #ifndef UTIL_H
2  #define UTIL_H
3
4  #include "config.h"
5
6  /* Convert from ascii hex digit to number */
7  uint8_t from_hex(uint8_t ch);
8  extern const uint8_t hex[16];
9  int32_t hex_to_u16(char x[4]);
10
11 /* Array length */
12 #define array_len(x) (sizeof(x)/sizeof(x[0]))
13
14 /* ISR disable/enable with saving of previous state */
15 #define __int_top() int __sr_save
16 #define __int_disable() __sr_save=SR; SRbits.IPL=7
17 #define __int_enable() SR=__sr_save
18 #define disable_int(x) do { \
19         __int_top(); \
20         __int_disable(); \
21         x; \
22         __int_enable(); } while(0)
23
24 /* Misc */
25 #define max(a,b)                                \
26         ({ typeof (a) _a = (a);                 \
27                 typeof (b) _b = (b);            \
28                 _a > _b ? _a : _b; })
29
30 #define min(a,b)                                \
31         ({ typeof (a) _a = (a);                 \
32                 typeof (b) _b = (b);            \
33                 _a < _b ? _a : _b; })
34
35 #define clamp(a,v,b) min(b,max(v,a))
36
37 #define abs(a)                                  \
38     ({ typeof (a) _a = (a);                     \
39        _a < 0 ? -_a : _a; })
40
41 #endif
```

Listing G-25: `firmware/util.c`: Miscellaneous firmware utilities, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `firmware/util.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include "config.h"
#include "util.h"

/* Convert from ASCII hex.  Returns
   the value, or 16 if it was space/newline, or
   32 if some other character. */
uint8_t from_hex(uint8_t ch)
{
        if(ch==' ' || ch=='\r' || ch=='\n')
                return 16;

        if(ch < '0')
                goto bad;
        if(ch <= '9')
                return ch - '0';
        ch |= 0x20;
        if(ch < 'a')
                goto bad;
        if(ch <= 'f')
                return ch - 'a' + 10;
 bad:
        return 32;
}

const uint8_t hex[16]={'0','1','2','3','4','5','6','7',
                       '8','9','a','b','c','d','e','f'};

/* Convert 4 ASCII hex digits into a 16-bit unsigned number.
   Returns the number, or -1 if there's an error */
int32_t hex_to_u16(char x[4])
{
        uint16_t v = 0;
        uint8_t t;
        int i;

        for (i = 0; i < 4; i++) {
                t = from_hex(x[i]);
                if (t >= 16)
                        return -1;
                v = (v << 4) | t;
        }
        return v;
}
```

G.2 PC Control Interface Source Code

This code communicates with and manages the Zoom NILM microcontroller via a USB connection. For calibration and testing, it also utilizes a GPIB connection to a Keithley 2002 MultiMeter and Keithley 2401 SourceMeter in order to measure and create voltages and currents. The following binaries are built from this code:

- `calibrate`: Performs calibration tasks such as writing specific values to the DAC, adjusting the DAC output such that the ADC input is centered, and performing sweeps to create the calibration data shown in Figure 5-7.

- `dactest`: Writes random values to the DAC while measuring it accurately, in order to ensure low-order bit stability and to create the LOOKUP table described in Section 5.4.3.4.

- `dctest`: Performs accuracy tests of the Zoom NILM system at randomized dc values, creating the data used in Figure 5-12.

- `read`: Reads realtime DAC, ADC, and clamp information from the Zoom NILM current sensor, used to capture data such as that used for Figure 5-11. This is the primary tool for performing data acquisition when running in normal operation, and does not utilize the Keithley hardware.

Listing G-26: `pc/calibrate.c`: Implementation of the `calibrate` tool.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/calibrate.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <errno.h>
#include <unistd.h>
#include <getopt.h>
#include <stdint.h>
#include <string.h>
#include <syslog.h>
#include <err.h>
#include <linux/serial.h>
#include <sys/signal.h>
#include "serial-util.h"
#include "gpib.h"
#include "zoom.h"
#include "math.h"

void zero(int zoom);
void write_dac(int zoom, int dac);
void single_sweep(int zoom);
void calibrate(int zoom, int gpib);
void sweep_ota(int zoom, int gpib, int ota_sweep_count);
void hold_ota(int zoom, int gpib, int ota_cmd);

#define info(x...) fprintf(stderr,x)

int g_quit = 0;
void handle_sig(int sig) { g_quit = 1; }
```

```c

int main(int argc, char *argv[])
{
        char *zoomdev=strdup("/dev/serial/by-id/usb-FTDI_FT232R_"
                             "USB_UART_A6007wc5-if00-port0");
        char *gpibdev=strdup("/dev/serial/by-id/usb-Prologix_Prologix_"
                             "GPIB-USB_Controller_PXQQY20G-if00-port0");
        int getopt_index;
        int zoom, gpib;
        int do_write = 0;
        int do_zero = 0;
        int write_cmd = 32768;
        int do_single_sweep = 0;
        int ota_opt = 0;
        int ota_sweep_count = 64;
        int ota_cmd = 32768;
        char *end;

        static struct option long_opts[] = {
                { "zoom-device", required_argument, NULL, 'Z' },
                { "gpib-device", required_argument, NULL, 'G' },
                { "write-dac", required_argument, NULL, 'w' },
                { "zero", no_argument, NULL, 'z' },
                { "single-sweep", no_argument, NULL, 's' },
                { "ota-sweep", required_argument, NULL, 'o' },
                { "ota-hold", required_argument, NULL, 'l' },
                { "help", no_argument, NULL, 'h' },
                { 0, 0, 0, 0}
        };
        int help=0;
        char c;

        while ((c = getopt_long(argc, argv, "Z:G:szw:o:l:h?",
                                long_opts, &getopt_index)) != -1) {
                switch(c)
                {
                case 'Z':
                        free(zoomdev);
                        zoomdev = strdup(optarg);
                        break;
                case 'G':
                        free(gpibdev);
                        gpibdev = strdup(optarg);
                        break;
                case 'w':
                        do_write = 1;
                        write_cmd = strtoul(optarg, &end, 0);
                        if (*end) {
                                fprintf(stderr, "bad number %s\n", optarg);
                                help = 1;
                        }
                        break;
                case 's':
                        do_single_sweep = 1;
                        break;
                case 'z':
                        do_zero = 1;
                        break;
                case 'o':
                        ota_opt = 1;
                        ota_sweep_count = strtoul(optarg, &end, 0);
                        if (*end) {
                                fprintf(stderr, "bad number %s\n", optarg);
                                help = 1;
                        }
                        break;
                case 'l':
```

```
                        ota_opt = 2;
                        ota_cmd = strtoul(optarg, &end, 0);
                        if (*end) {
                                fprintf(stderr, "bad number %s\n", optarg);
                                help = 1;
                        }
                        break;
                case 'h':

                case '?':
                default:
                        help = 1;
                        break;
                }
        }

        if (help) {
                fprintf(stderr, "Zoom Nilm Calibration Tool\n");
                fprintf(stderr, "usage: %s [options]\n\n", *argv);
                fprintf(stderr, "  -Z,  --zoom-device %-14s "
                        "zoom NILM serial port\n", "/dev/xxx" /*zoomdev*/);
                fprintf(stderr, "  -G,  --gpib-device %-14s "
                        "GPIB serial port\n", "/dev/xxx" /*gpibdev*/);
                fprintf(stderr, "  -w,  --write-dac %-14d   "
                        "write one value to the DAC constantly\n", write_cmd);
                fprintf(stderr, "  -s,  --single-sweep      "
                        "do a single sweep on the PIC\n");
                fprintf(stderr, "  -z,  --zero              "
                        "set DAC value such that ADC input is centered\n");
                fprintf(stderr, "  -o,  --ota-sweep %-14d   "
                        "sweep OTA\n", ota_sweep_count);
                fprintf(stderr, "  -l,  --ota-hold %-14d    "
                        "hold OTA constant\n", ota_cmd);
                fprintf(stderr, "  -h,  --help              "
                        "this help\n");
                return 1;
        }

        signal(SIGINT, handle_sig);

        if ((zoom = serial_open(zoomdev, 115200)) == -1)
                err(1, "failed to open zoom device %s", zoomdev);

        if (do_write) {
                write_dac(zoom, write_cmd);
                close(zoom);
                return 0;
        }

        if (do_zero) {
                zero(zoom);
                close(zoom);
                return 0;
        }

        if (do_single_sweep) {
                single_sweep(zoom);
                close(zoom);
                return 0;
        }

        if ((gpib = serial_open(gpibdev, 9600)) == -1)
                err(1, "failed to open gpib device %s", gpibdev);

        switch (ota_opt) {
        case 1:
                sweep_ota(zoom, gpib, ota_sweep_count);
```

```
                break;
        case 2:
                hold_ota(zoom, gpib, ota_cmd);
                break;
        default:
                calibrate(zoom, gpib);
                break;
        }

        close(zoom);
        close(gpib);
        return 0;
}

void calibrate(int zoom, int gpib)
{
        double idesired, iactual;
        int i;
        int zero;
        int r = 0;
        int dac[ZOOM_SWEEP_COUNT];
        int adc[ZOOM_SWEEP_COUNT];

        info("Initializing Zoom NILM\n");
        if (zoom_init(zoom) < 0) goto fail;

        info("Zeroing\n");
        if (zoom_zero_start(zoom) < 0) goto fail;

        info("Initializing GPIB\n");
        if (gpib_init(gpib) < 0) goto fail;

        info("Initializing Keithley\n");
        if (gpib_addr(gpib, 24) < 0) goto fail;
        if (keithley_init(gpib) < 0) goto fail;
        if (keithley_current(gpib, 0) < 0) goto fail;
        if (isnan(keithley_read(gpib))) goto fail;

        info("Stop zeroing\n");
        if (zoom_zero_stop(zoom) < 0) goto fail;

        info("Sweeping\n");
        for (idesired = -1.0; idesired <= 1.0 && !g_quit; idesired += 0.10) {
//      for (idesired = -0.2; idesired <= 0.2 && !g_quit; idesired += 0.02) {
                info("Zeroing\n");
                if (zoom_zero_start(zoom) < 0) goto fail;

                info("Setting current: %.8f\n", idesired);
                keithley_current(gpib, idesired);
                usleep(100000);
                iactual = keithley_read(gpib);
                info("Actual current: %.8f\n", iactual);

                info("Stop zeroing\n");
                if ((zero = zoom_zero_stop(zoom)) < 0) goto fail;
                info("DAC zero point = %d\n", zero);

                info("Sweeping...\n");
                if ((r = zoom_sweep(zoom, dac, adc)) < 0) goto fail;

                info("Done\n");
                for (i = 0; i < ZOOM_SWEEP_COUNT; i++) {
                        printf("%.8f %d %d\n", iactual, dac[i], adc[i]);
                }
        }

safecleanup:
```

```
        zoom_zero_start(zoom);
        keithley_off(gpib);
        usleep(50000);
        zoom_zero_stop(zoom);
        return;

fail:
        info("Failed (code %d)\n", r);
        goto safecleanup;
}

void write_dac(int zoom, int dac)
{
        char buf[128];

        if (zoom_init(zoom) < 0)
                errx(1, "init failed");

        if (dac < 0)
                dac = 0;
        if (dac > 65535)
                dac = 65535;
        while (!g_quit) {
                sprintf(buf, "v%04x", dac);
                if (safewrite(zoom, buf, 5) != 5)
                        errx(1, "write failed");
                if (fdgets(buf, 128, zoom, 1000) == NULL)
                        errx(1, "read timeout");
                chomp(buf);
                printf("%s\n", buf);
        }
}

void single_sweep(int zoom)
{
        int i, r;
        int dac[ZOOM_SWEEP_COUNT];
        int adc[ZOOM_SWEEP_COUNT];

        info("Initializing Zoom NILM\n");
        if (zoom_init_nozero(zoom) < 0) goto fail;

        info("Sweeping\n");
        if ((r = zoom_sweep(zoom, dac, adc)) < 0) goto fail;

        info("Done\n");
        for (i = 0; i < ZOOM_SWEEP_COUNT; i++) {
                printf("%d %d\n", dac[i], adc[i]);
        }

        return;

fail:
        info("Failed (code %d)\n", r);
        return;
}

void zero(int zoom)
{
        int r = 0;
        int dac;

        info("Initializing Zoom NILM\n");
        if (zoom_init(zoom) < 0) goto fail;

        info("Starting zeroing, ^C to stop\n");
        if (zoom_zero_start(zoom) < 0) goto fail;
```

```
297
298         while (!g_quit)
299                 usleep(100000);
300
301         info("Stop zeroing\n");
302         if ((dac = zoom_zero_stop(zoom)) < 0) goto fail;
303         info("DAC zero point = 0x%04x %d\n", dac, dac);
304
305         info("Done\n");
306         return;
307
308 fail:
309         info("Failed (code %d)\n", r);
310         return;
311 }
312
313 void sweep_ota(int zoom, int gpib, int ota_sweep_count)
314 {
315         double iactual;
316         int i,j;
317         int r = 0;
318         char buf[128];
319         char zcmd[128];
320         int mycmd = 0;
321         int cmd_inc = 4096;
322
323         info("Initializing Zoom NILM\n");
324         if (zoom_init(zoom) < 0) goto fail;
325
326         info("Initializing GPIB\n");
327         if (gpib_init(gpib) < 0) goto fail;
328
329         info("Initializing Keithley 2002 Multimeter\n");
330         if (gpib_addr(gpib, 23) < 0) goto fail;
331         if (keithley2002_init(gpib) < 0) goto fail;
332         if (isnan(keithley2002_read(gpib))) goto fail;
333
334         buf[0] = '0';
335         if (safewrite(zoom, buf, 1) != 1)
336                 errx(1, "write failed");
337         if (fdgets(buf, 128, zoom, 1000) == NULL)
338                 errx(1, "read timeout");
339         drain(zoom);
340
341
342
343         info("Sweeping OTA\n");
344         for (i = 0; i <= 16 && !g_quit; i++){
345
346                 for(j = -2; j <= 2; j++){
347                         if(mycmd+j > 65535 || mycmd+j < 0)
348                                 continue;
349
350                         if(sprintf(zcmd,"v%.4x",(mycmd +j)) < 5)
351                                 errx(1, "fail hex conversion");
352
353                         if (safewrite(zoom, zcmd, 5) != 5)
354                                 errx(1, "write failed");
355                         if (fdgets(buf, 128, zoom, 1000) == NULL)
356                                 errx(1, "read timeout");
357
358                         usleep(100);
359
360                         iactual = keithley2002_read(gpib);
361
362                         printf("%d %.8f\n",(int)(mycmd + j), iactual);
363                 }
```

```
364
365                 mycmd += cmd_inc;
366
367         }
368
369         // return to zero
370         buf[0] = '0';
371         if (safewrite(zoom, buf, 1) != 1)
372                 errx(1, "write failed");
373         if (fdgets(buf, 128, zoom, 1000) == NULL)
374                 errx(1, "read timeout");
375
376
377
378 safecleanup:
379         return;
380
381 fail:
382         info("Failed (code %d)\n", r);
383         goto safecleanup;
384 }
385
386
387 void hold_ota(int zoom, int gpib, int ota_cmd)
388 {
389         double iactual;
390         double tx;
391         int i;
392         int r = 0;
393         char buf[128];
394         char zcmd[128];
395
396         info("Initializing Zoom NILM\n");
397         if (zoom_init(zoom) < 0) goto fail;
398
399         info("Initializing GPIB\n");
400         if (gpib_init(gpib) < 0) goto fail;
401
402         info("Initializing Keithley 2002 Multimeter\n");
403         if (gpib_addr(gpib, 23) < 0) goto fail;
404         if (keithley2002_init2(gpib) < 0) goto fail;
405         if (isnan(keithley2002_read(gpib))) goto fail;
406
407         buf[0] = '0';
408         if (safewrite(zoom, buf, 1) != 1)
409                 errx(1, "write failed");
410         if (fdgets(buf, 128, zoom, 1000) == NULL)
411                 errx(1, "read timeout");
412         drain(zoom);
413
414         if(ota_cmd > 65535 || ota_cmd < 0)
415                 errx(1, "ota command out-of-range");
416
417         if(sprintf(zcmd,"v%.4x",(ota_cmd)) < 5)
418                 errx(1, "fail hex conversion");
419
420         if (safewrite(zoom, zcmd, 5) != 5)
421                 errx(1, "write failed");
422         if (fdgets(buf, 128, zoom, 1000) == NULL)
423                 errx(1, "read timeout");
424
425         usleep(100);
426
427         info("Holding OTA\n");
428         for (i = 0; i <= 500 && !g_quit; i++){
429
430                 iactual = keithley2002_read2(gpib, &tx);
```

```
431
432                 printf("%.12f %.12f\n", tx, iactual);
433
434         }
435
436         // return to zero
437         buf[0] = '0';
438         if (safewrite(zoom, buf, 1) != 1)
439                 errx(1, "write failed");
440         if (fdgets(buf, 128, zoom, 1000) == NULL)
441                 errx(1, "read timeout");
442
443
444
445 safecleanup:
446         return;
447
448 fail:
449         info("Failed (code %d)\n", r);
450         goto safecleanup;
451 }
```

Listing G-27: `pc/dactest.c`: Implementation of the `dactest` tool.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/dactest.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  /* Test DAC for stability etc */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <sys/types.h>
6  #include <errno.h>
7  #include <unistd.h>
8  #include <getopt.h>
9  #include <stdint.h>
10 #include <string.h>
11 #include <syslog.h>
12 #include <err.h>
13 #include <linux/serial.h>
14 #include <sys/signal.h>
15 #include <sys/time.h>
16 #include "serial-util.h"
17 #include "gpib.h"
18 #include "zoom.h"
19 #include <math.h>
20 #include "mt19937ar.h"
21
22 void write_random(int zoom, int gpib);
23
24 #define info(x...) fprintf(stderr,x)
25
26 int g_quit = 0;
27 void handle_sig(int sig) { g_quit = 1; }
28
29 int main(int argc, char *argv[])
30 {
31         char *zoomdev=strdup("/dev/serial/by-id/usb-FTDI_"
32                              "FT232R_USB_UART_A6007wc5-if00-port0");
33         char *gpibdev=strdup("/dev/serial/by-id/usb-Prologix_"
34                              "Prologix_GPIB-USB_Controller_PXQQY20G-if00-port0");
35         int getopt_index;
36         int zoom, gpib;
```

```
        unsigned long seed = 1337;

        static struct option long_opts[] = {
                { "zoom-device", required_argument, NULL, 'Z' },
                { "gpib-device", required_argument, NULL, 'G' },
                { "seed", required_argument, NULL, 's' },
                { "help", no_argument, NULL, 'h' },
                { 0, 0, 0, 0}
        };
        int help=0;
        char c;

        while ((c = getopt_long(argc, argv, "Z:G:s:h?",
                                long_opts, &getopt_index)) != -1) {
                switch(c)
                {
                case 'Z':
                        free(zoomdev);
                        zoomdev = strdup(optarg);
                        break;
                case 'G':
                        free(gpibdev);
                        gpibdev = strdup(optarg);
                        break;
                case 's':
                        seed = atol(optarg);
                        if(seed == 0)
                                errx(1, "invalid seed: %s", optarg);
                        break;
                case 'h':
                case '?':
                default:
                        help = 1;
                        break;
                }
        }

        if (help) {
                fprintf(stderr, "Zoom Nilm DC Test\n");
                fprintf(stderr, "usage: %s [options]\n\n", *argv);
                fprintf(stderr, "  -Z,  --zoom-device %-14s "
                        "zoom NILM serial port\n", "/dev/xxx" /*zoomdev*/);
                fprintf(stderr, "  -G,  --gpib-device %-14s "
                        "GPIB serial port\n", "/dev/xxx" /*gpibdev*/);
                fprintf(stderr, "  -s,  --seed %-16ld random seed\n", seed);
                fprintf(stderr, "  -h,  --help                      "
                        "this help\n");
                return 1;
        }

        signal(SIGINT, handle_sig);

        info("Initializing twister with seed %ld\n", seed);
        init_genrand(seed);

        info("Opening Zoom NILM device %s\n", zoomdev);
        if ((zoom = serial_open(zoomdev, 115200)) == -1)
                err(1, "failed to open zoom device %s", zoomdev);

        info("Opening GPIB device %s\n", gpibdev);
        if ((gpib = serial_open(gpibdev, 9600)) == -1)
                err(1, "failed to open gpib device %s", gpibdev);

        /* do it */
        write_random(zoom, gpib);

        close(zoom);
```

```
104         close(gpib);
105         return 0;
106 }
107
108 void write_random(int zoom, int gpib)
109 {
110         int dac;
111         double meas;
112         struct timeval now;
113
114         info("Initializing Zoom NILM\n");
115         if (zoom_init_nozero(zoom) < 0) goto fail;
116
117         info("Initializing GPIB\n");
118         if (gpib_init(gpib) < 0) goto fail;
119
120         info("Initializing Keithley\n");
121         if (gpib_addr(gpib, 23) < 0) goto fail;
122         if (keithley2002_init_volts(gpib) < 0) goto fail;
123         if (isnan(keithley2002_read(gpib))) goto fail;
124
125         info("Running\n");
126         while (!g_quit) {
127                 dac = genrand_int32() & 0x3ff;
128                 zoom_write_dac(zoom, dac);
129                 gettimeofday(&now, NULL);
130                 meas = keithley2002_read(gpib);
131                 printf("%ld.%06ld %d %.12f\n", now.tv_sec, now.tv_usec,
132                        dac, meas);
133                 fflush(stdout);
134         }
135
136 safecleanup:
137         return;
138
139 fail:
140         info("Failed\n");
141         goto safecleanup;
142 }
```

Listing G-28: `pc/dctest.c`: Implementation of the `dctest` tool.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/dctest.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <errno.h>
5  #include <unistd.h>
6  #include <getopt.h>
7  #include <stdint.h>
8  #include <string.h>
9  #include <syslog.h>
10 #include <err.h>
11 #include <linux/serial.h>
12 #include <sys/signal.h>
13 #include "serial-util.h"
14 #include "gpib.h"
15 #include "zoom.h"
16 #include "math.h"
17 #include <pthread.h>
18 #include <ctype.h>
```

```
#include "mt19937ar.h"

#define info(x...) fprintf(stderr,x)

static void dctest(int zoom, int gpib);

int g_quit = 0;
static void handle_sig(int sig) { g_quit = 1; }

int main(int argc, char *argv[])
{
        char *zoomdev=strdup("/dev/serial/by-id/usb-FTDI_"
                             "FT232R_USB_UART_A6007wc5-if00-port0");
        char *gpibdev=strdup("/dev/serial/by-id/usb-Prologix_"
                             "Prologix_GPIB-USB_Controller_PXQQY20G-if00-port0");
        int rate=500000;
        unsigned long seed = 1337;
        int getopt_index;
        int zoom, gpib;

        static struct option long_opts[] = {
                { "zoom-device", required_argument, NULL, 'Z' },
                { "gpib-device", required_argument, NULL, 'G' },
                { "rate", required_argument, NULL, 'r' },
                { "seed", required_argument, NULL, 's' },
                { "help", no_argument, NULL, 'h' },
                { 0, 0, 0, 0}
        };
        int help=0;
        char c;

        while ((c = getopt_long(argc, argv, "Z:G:r:s:h?",
                                long_opts, &getopt_index)) != -1) {
                switch(c)
                {
                case 'Z':
                        free(zoomdev);
                        zoomdev = strdup(optarg);
                        break;
                case 'G':
                        free(gpibdev);
                        gpibdev = strdup(optarg);
                        break;
                case 'r':
                        rate = atoi(optarg);
                        if(rate == 0)
                                errx(1, "invalid rate: %s", optarg);
                        break;
                case 's':
                        seed = atol(optarg);
                        if(seed == 0)
                                errx(1, "invalid seed: %s", optarg);
                        break;
                case 'h':
                case '?':
                default:
                        help = 1;
                        break;
                }
        }

        if (help) {
                fprintf(stderr, "Zoom Nilm DC Test Tool\n");
                fprintf(stderr, "usage: %s [options]\n\n", *argv);
                fprintf(stderr, "  -Z,  --zoom-device %-9s "
                        "zoom NILM serial port\n", "/dev/xxx");
                fprintf(stderr, "  -G,  --gpib-device %-9s "
```

```
                        "GPIB serial port\n", "/dev/xxx");
                fprintf(stderr, "  -r,  --rate %-16d baud rate\n", rate);
                fprintf(stderr, "  -s,  --seed %-16ld random seed\n", seed);
                fprintf(stderr, "  -h,  --help                 this help\n");
                return 1;
        }

        signal(SIGINT, handle_sig);

        info("Initializing twister with seed %ld\n", seed);
        init_genrand(seed);

        /* open devices */
        info("Opening Zoom NILM device %s\n", zoomdev);
        if ((zoom = serial_open(zoomdev, rate)) == -1)
                err(1, "failed to open zoom device %s", zoomdev);

        info("Opening GPIB device %s\n", gpibdev);
        if ((gpib = serial_open(gpibdev, 9600)) == -1)
                err(1, "failed to open gpib device %s", gpibdev);

        /* do the dc test */
        dctest(zoom, gpib);

        close(zoom);
        close(gpib);
        return 0;
}

struct keithley_t {
        double desired;
        double actual;
        int stable;
};

struct threadinfo_t {
        int quit_flag;
        int fd;
        pthread_mutex_t mutex;
        float calibration;
        struct keithley_t k;
};

int process_adc_dac(const uint8_t *buf, struct threadinfo_t *ti)
{
        uint16_t dac, tmp;
        int16_t adc;
        int overflow;
        double idesired, iactual;
        double calib;
        int stable;

        /* data OK? */
        if ((buf[0] & 0xC0) != 0) return 0;

        /* extract */
        overflow = (buf[0] & 0x10) ? 1 : 0;
        tmp = ((buf[0] & 0x0F) << 8) | buf[1];

        /* sign-extend ADC value */
        if (tmp & 0x0800)
                tmp |= 0xF000;
        else
                tmp &= ~0xF000;
        adc = (int16_t)tmp;

        dac = (buf[2] << 8) | buf[3];
```

```
153
154         /* get locked data */
155         pthread_mutex_lock(&ti->mutex);
156         idesired = ti->k.desired;
157         iactual = ti->k.actual;
158         stable = ti->k.stable;
159         calib = ti->calibration;
160         pthread_mutex_unlock(&ti->mutex);
161
162         /* write it out */
163         printf("%d %.12f %.12f %.8f %5d %5d %d\n",
164                stable,
165                idesired,
166                iactual,
167                calib,
168                dac,
169                adc,
170                overflow);
171
172         return 1;
173 }
174
175 static int process_calibration(const uint8_t *buf, struct threadinfo_t *ti)
176 {
177         float f = *(float *)buf;
178         pthread_mutex_lock(&ti->mutex);
179         ti->calibration = f;
180         pthread_mutex_unlock(&ti->mutex);
181         info("New calibration value: %.8f\n", f);
182         return 1;
183 }
184
185 static int process(const uint8_t *buf, int len, struct threadinfo_t *ti)
186 {
187         int n = 0;
188
189         /* Process blocks */
190  retry:
191         for (; (n + 5) <= len; buf += 5, n += 5) {
192                 int ok = 0;
193                 switch (buf[0]) {
194                 case 0xA0:
195                         if (process_adc_dac(buf + 1, ti)) ok = 1;
196                         break;
197                 case 0xA1:
198                         if (process_calibration(buf + 1, ti)) ok = 1;
199                         break;
200                 default:
201                         break;
202                 }
203                 if (!ok) {
204                         /* badly formed data; eat one byte and retry */
205                         info("throwing away 0x%02x '%c'\n", buf[0],
206                              isprint(buf[0]) ? buf[0] : '.');
207                         buf++;
208                         n++;
209                         goto retry;
210                 }
211         }
212
213         return n;
214 }
215
216 static void *read_data(void *arg)
217 {
218         struct threadinfo_t *ti = (struct threadinfo_t *)arg;
219         char buf[1024];
```

```
220         int len;
221
222         /* read data in a loop.  Use saferead_timeout here so we can
223            notice quit_flag before too long. */
224         len = 0;
225         while (!ti->quit_flag) {
226                 int processed, n;
227                 n = saferead_timeout(ti->fd,
228                                      buf + len,
229                                      sizeof(buf) - len,
230                                      1000);
231                 if (n < 0)
232                         err(1, "read");
233                 if (n == 0)
234                         continue;
235                 len += n;
236                 processed = process((uint8_t *) buf, len, ti);
237                 memmove(buf, buf + processed, len - processed);
238                 len -= processed;
239         }
240         info("read thread quitting\n");
241         return NULL;
242 }
243
244 /* change keithley and update keithley_t structure with locking */
245 static int keithley_change(int gpib, double desired, struct threadinfo_t *ti)
246 {
247         double actual;
248
249         pthread_mutex_lock(&ti->mutex);
250         ti->k.stable = 0;
251         ti->k.desired = desired;
252         pthread_mutex_unlock(&ti->mutex);
253
254         if (keithley_current(gpib, desired) < 0)
255                 return -1;
256
257         actual = keithley_read(gpib);
258         if (isnan(actual))
259                 return -1;
260
261         pthread_mutex_lock(&ti->mutex);
262         ti->k.actual = actual;
263         ti->k.stable = 1;
264         pthread_mutex_unlock(&ti->mutex);
265         return 0;
266 }
267
268 static double genrand(double min, double max)
269 {
270         double x;
271         x = genrand_real1();    /* [0, 1] */
272         x *= (max - min);       /* [0, max-min] */
273         x += min;               /* [min, max] */
274         return x;
275 }
276
277 static void dctest(int zoom, int gpib)
278 {
279         pthread_t thread;
280         struct threadinfo_t ti;
281         double tmp;
282         int i;
283
284         /* Do a calibration with Keithley off */
285         info("Triggering calibration\n");
286         zoomrun_trigger_calibrate(zoom);
```

```
287         usleep(500000);
288
289         /* Init Keithley */
290         info("Initializing GPIB\n");
291         if (gpib_init(gpib) < 0) { info("failed\n"); goto out1; }
292
293         info("Initializing Keithley\n");
294         if (gpib_addr(gpib, 24) < 0) { info("failed\n"); goto out1; }
295         if (keithley_init(gpib) < 0) { info("failed\n"); goto out2; }
296         if (keithley_current(gpib, 0) < 0) { info("failed\n"); goto out2; }
297         if (isnan(keithley_read(gpib))) { info("failed\n"); goto out2; }
298
299         /* Start the thread that reads and dumps data */
300         info("Spawning thread\n");
301         if (pthread_mutex_init(&ti.mutex, NULL) != 0) {
302                 info("failed\n");
303                 goto out2;
304         }
305         ti.calibration = 0.0;
306         ti.k.desired = 0;
307         ti.k.actual = 0;
308         ti.k.stable = 0;
309         ti.quit_flag = 0;
310         ti.fd = zoom;
311         drain_timeout(zoom, 0);
312         if (pthread_create(&thread, NULL, read_data, &ti) != 0) {
313                 info("failed\n");
314                 goto out2;
315         }
316
317         /* Do another calibration now, and verify it works */
318         info("Triggering calibration\n");
319         pthread_mutex_lock(&ti.mutex);
320         ti.calibration = 0.0;
321         pthread_mutex_unlock(&ti.mutex);
322         if (keithley_change(gpib, 0.0, &ti) < 0) { info("failed\n"); goto out3; }
323         zoomrun_trigger_calibrate(zoom);
324         for (i = 0; i < 100; i++) {
325                 usleep(50000);
326                 pthread_mutex_lock(&ti.mutex);
327                 tmp = ti.calibration;
328                 pthread_mutex_unlock(&ti.mutex);
329                 if (tmp != 0.0)
330                         break;
331         }
332         if (tmp == 0.0) {
333                 info("Invalid calibration data: is zoom NILM working?\n");
334                 goto out3;
335         }
336
337         info("Running\n");
338         /* Change Keithley values */
339         while (!g_quit) {
340
341 #define DELAY 100000    /* after keithley settles, in us */
342 #define STEPS 10        /* how many small steps to take */
343
344 #define K_MIN -0.5      /* keithley min, amps */
345 #define K_MAX 0.5       /* keithley max, amps */
346 #define STEPSIZE 0.03   /* max size of small step, in amps */
347
348                 /* Choose any value within the Keithley range */
349                 double desired = genrand(K_MIN, K_MAX);
350                 info("Big step: %.12f\n", desired);
351                 if (keithley_change(gpib, desired, &ti) < 0) {
352                         info("error setting keithley\n");
353                         break;
```

```
354                 }
355                 usleep(DELAY);
356
357                 /* Choose a few more values nearby */
358                 for (i = 0; i < STEPS && !g_quit; i++) {
359                         desired += genrand(-STEPSIZE/2, STEPSIZE/2);
360                         if (desired < -1.0)
361                                 desired = -1.0;
362                         if (desired > 1.0)
363                                 desired = 1.0;
364                         if (keithley_change(gpib, desired, &ti) < 0) {
365                                 info("error setting keithley\n");
366                                 break;
367                         }
368                         usleep(DELAY);
369                 }
370
371                 /* one extra delay just to separate things a bit */
372                 usleep(DELAY);
373         }
374         info("main thread quitting\n");
375
376 out3:
377         ti.quit_flag = 1;
378         pthread_join(thread, NULL);
379 out2:
380         keithley_off(gpib);
381 out1:
382         return;
383 }
```

Listing G-29: `pc/read.c`: Implementation of the `read` tool.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/read.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <errno.h>
5  #include <unistd.h>
6  #include <getopt.h>
7  #include <stdint.h>
8  #include <syslog.h>
9  #include <err.h>
10 #include <linux/serial.h>
11 #include "serial-util.h"
12 #include <string.h>
13 #include <ctype.h>
14
15 int hex = 0;
16 int dec = 0;
17 int screen = 0;
18 int unprocessed = 0;
19
20 int process(const uint8_t *buf, int len);
21
22 int main(int argc, char *argv[])
23 {
24         char *device=strdup("/dev/serial/by-id/usb-FTDI_FT232R_"
25                             "USB_UART_A6007wc5-if00-port0");
26         int rate=500000;
27         int fd;
```

```
        int getopt_index;
        char buf[1024];
        int len;
        int calibrate = 1;

        static struct option long_opts[] = {
                { "device", required_argument, NULL, 'd' },
                { "rate", required_argument, NULL, 'r' },
                { "hex", no_argument, NULL, 'x' },
                { "dec", no_argument, NULL, 'D' },
                { "unprocessed", no_argument, NULL, 'u' },
                { "no-calibrate", no_argument, NULL, 'n' },
                { "screen", no_argument, NULL, 's' },
                { "help", no_argument, NULL, 'h' },
                { 0, 0, 0, 0 }
        };
        int help=0;
        char c;

        while ((c = getopt_long(argc, argv, "d:r:xDunsh?",
                                long_opts, &getopt_index)) != -1) {
                switch(c)
                {
                case 'd':
                        free(device);
                        device = strdup(optarg);
                        break;
                case 'r':
                        rate = atoi(optarg);
                        if(rate == 0)
                                errx(1, "invalid rate: %s",optarg);
                        break;
                case 'x':
                        hex = 1;
                        break;
                case 'D':
                        dec = 1;
                        break;
                case 'u':
                        unprocessed = 1;
                        break;
                case 'n':
                        calibrate = 0;
                        break;
                case 's':
                        screen = 1;
                        break;
                case 'h':
                case '?':
                default:
                        help = 1;
                        break;
                }
        }

        if (help) {
                fprintf(stderr, "Zoom Nilm Client\n");
                fprintf(stderr, "usage: %s [options]\n\n", *argv);
                fprintf(stderr, "  -d,  --device %-14s serial port\n", device);
                fprintf(stderr, "  -r,  --rate %-16d baud rate\n", rate);
                fprintf(stderr, "  -x,  --hex                    hex out\n");
                fprintf(stderr, "  -D,  --dec                    dec out\n");
                fprintf(stderr, "  -u,  --unprocessed            "
                        "dump raw unprocessed data\n");
                fprintf(stderr, "  -n,  --no-calibrate           "
                        "skip calibration routine\n");
                fprintf(stderr, "  -s,  --screen                 "
```

```
                        "send \\r instead of \\n\n");
                fprintf(stderr, "  -h,  --help                  this cruft\n");
                return 1;
        }

        if ((fd = serial_open(device, rate)) == -1)
                err(1, "serial_open failed for %s", device);

        if (calibrate) {
                fprintf(stderr, "performing calibration\n");
                char c = 'c';
                write(fd, &c, 1);
                drain(fd);
        }

        len = 0;
        while (1) {
                int processed, n;
                n = read(fd, buf + len, sizeof(buf) - len);
                if (n <= 0)
                        err(1, "read");
                len += n;
                processed = process((uint8_t *) buf, len);
                memmove(buf, buf + processed, len - processed);
                len -= processed;
        }

        return 1;
}

int process_adc_dac(const uint8_t *buf)
{
        uint16_t dac, tmp;
        int16_t adc;
        int overflow;

        /* data OK? */
        if ((buf[0] & 0xC0) != 0)
                return 0;

        /* extract */

        if (buf[0] & 0x10)
                overflow = 1;
        else
                overflow = 0;

        tmp = ((buf[0] & 0x0F) << 8) | buf[1];
        /* sign-extend ADC value */
        if (tmp & 0x0800)
                tmp |= 0xF000;
        else
                tmp &= ~0xF000;
        adc = (int16_t)tmp;

        dac = (buf[2] << 8) | buf[3];

        /* send it out */
        if (hex) {
                printf("%04x %03x", dac, adc & 0x0FFF);
        } else if (dec) {
                printf("%d %d", dac, adc);
        } else {
                printf("DAC: %5d   ADC: % 5d  Total: xxx", dac, adc);
                if (overflow)
                        printf("   **** adc may have clamped");
        }
```

```
162
163         if (screen)
164                 printf("\033[K\r");
165         else
166                 printf("\n");
167
168         return 1;
169 }
170
171 int process_calibration(const uint8_t *buf)
172 {
173         float f = *(float *)buf;
174
175         fprintf(stderr, "got calibration value: %f\n", f);
176
177         return 1;
178 }
179
180 int process(const uint8_t *buf, int len)
181 {
182         int n = 0;
183
184         if (unprocessed) {
185                 n = write(fileno(stdout), buf, len);
186                 if (n >= 0)
187                         return n;
188                 return 0;
189         }
190
191         /* Process blocks */
192  retry:
193         for (; (n + 5) <= len; buf += 5, n += 5) {
194                 int ok = 0;
195                 switch (buf[0]) {
196                 case 0xA0:
197                         if (process_adc_dac(buf + 1)) ok = 1;
198                         break;
199                 case 0xA1:
200                         if (process_calibration(buf + 1)) ok = 1;
201                         break;
202                 default:
203                         break;
204                 }
205                 if (!ok) {
206                         /* badly formed data; eat one byte and retry */
207                         fprintf(stderr,"throwing away 0x%02x '%c'\n",
208                                 buf[0], isprint(buf[0]) ? buf[0] : '.');
209                         buf++;
210                         n++;
211                         goto retry;
212                 }
213         }
214
215         return n;
216 }
```

Listing G-30: `pc/zoom.h`: High level routines for interfacing with Zoom NILM debug mode, header file. These routines communicate with the Zoom NILM running in debug mode, and can trigger the firmware to perform zeroing, sweep output, write DAC values, etc.

Git repository: https://git.jim.sh/jim/lees/zoom.git
Filename: pc/zoom.h
Revision: e28af0bb6747b0631f28ffd09813c45105e6906c

```
#ifndef ZOOM_H
#define ZOOM_H

#define ZOOM_SWEEP_COUNT 4000

/* debug mode */
int zoom_init_real(int fd, int dozero);
#define zoom_init(fd) zoom_init_real(fd, 1)
#define zoom_init_nozero(fd) zoom_init_real(fd, 0)
int zoom_zero_start(int fd);
int zoom_zero_stop(int fd);
int zoom_sweep(int fd, int dac[ZOOM_SWEEP_COUNT], int adc[ZOOM_SWEEP_COUNT]);
int zoom_write_dac(int fd, int dac);

/* run mode */
void zoomrun_trigger_calibrate(int fd);

#endif
```

Listing G-31: `pc/zoom.c`: High level routines for interfacing with Zoom NILM debug mode, implementation.

Git repository: https://git.jim.sh/jim/lees/zoom.git
Filename: pc/zoom.c
Revision: e28af0bb6747b0631f28ffd09813c45105e6906c

```
#include <stdio.h>
#include <string.h>
#ifndef __USE_ISOC99
#define __USE_ISOC99
#endif
#include <math.h>
#include "zoom.h"
#include "serial-util.h"

#define zputs(s) do { if (safewrite(fd, s, strlen(s)) != strlen(s)) \
                        return -1; } while(0)
#define zputc(ch) do { const char ____c = ch; \
                if (safewrite(fd, &____c, 1) != 1) return -1; } while(0)

static int last_dac, last_adc;

static int verify_prompt(int fd)
{
        char s[128];
        int dac1, dac2, adc1, adc2;
        if (fdgets(s, 128, fd, 1000) == NULL)
                return -1;
        chomp(s);
        if (sscanf(s, "%x %d  %x %d", &dac1, &dac2, &adc1, &adc2) != 4)
                return -1;
//      if (dac1 != dac2 || adc1 != adc2)
```

```
        if (adc1 != adc2)
                return -1;
        last_dac = dac1;
        last_adc = adc1;
        return 0;
}

int zoom_init_real(int fd, int dozero)
{
        zputs(dozero ? "00000000" : "        ");
        drain(fd);
        zputc(dozero ? '0' : ' ');
        if (verify_prompt(fd) < 0)
                return -1;
        return 0;
}

int zoom_zero_start(int fd)
{
        char s[128];
        zputc('z');
        if (fdgets(s, 128, fd, 1000) == NULL)
                return -1;
        chomp(s);
        if (strcmp(s, "zeroing input...") != 0)
                return -1;
        return 0;
}

int zoom_zero_stop(int fd)
{
        zputc(' ');
        if (verify_prompt(fd) < 0)
                return -1;
        return last_dac;
}

int zoom_sweep(int fd, int dac[ZOOM_SWEEP_COUNT], int adc[ZOOM_SWEEP_COUNT])
{
        char s[128];
        int i;
        char c;
        zputc('s');
        if (fdgets(s, 128, fd, 1000) == NULL)
                return -1;
        if (strncmp(s, "sweep around", 12) != 0)
                return -1;
        for (i = 0; i < ZOOM_SWEEP_COUNT; i++) {
                if (fdgets(s, 128, fd, 1000) == NULL)
                        return -1;
                chomp(s);
                if (sscanf(s, "%d %d%c", &dac[i], &adc[i], &c) != 2)
                        return -2;
        }
        if (verify_prompt(fd) < 0)
                return -3;
        return 0;
}

int zoom_write_dac(int fd, int dac)
{
        char s[128];

        sprintf(s, "v%04x", dac);
        zputs(s);
        if (verify_prompt(fd) < 0)
                return -1;
```

```
94          return last_dac;
95  }
96
97  /* Run mode: */
98
99  void zoomrun_trigger_calibrate(int fd)
100 {
101         char c = 'c';
102         write(fd, &c, 1);
103         drain_timeout(fd, 0);
104 }
```

Listing G-32: `pc/mt19937ar.h`: Mersenne twister pseudo random number generator. This is used for the randomized dc accuracy tests.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/mt19937ar.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1   /* This header file provides prototypes for the Mersenne Twister
2      random number generator.  The corresponding mt19937.c is also
3      required and can be obtained from:
4        http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html
5   */
6   void init_genrand(unsigned long s);
7   void init_by_array(unsigned long init_key[], int key_length);
8   unsigned long genrand_int32(void);
9   long genrand_int31(void);
10  double genrand_real1(void);
11  double genrand_real2(void);
12  double genrand_real3(void);
13  double genrand_res53(void);
```

Listing G-33: `pc/serial-util.h`: Serial interface driver, header file. This code includes support for configuring the FT232 USB to serial interface chip for the custom 500000 baud communication rate used by the Zoom NILM in “run” mode.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/serial-util.h`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1   /*
2    * Serial I/O helper routines
3    *
4    * Jim Paris <jim@jtan.com>
5    * $Id$
6    */
7   #ifndef SERIAL_UTIL_H
8   #define SERIAL_UTIL_H
9
10  #include <unistd.h>
11
12  /* Open serial port in raw mode, with custom baudrate if necessary */
13  int serial_open(const char *device, int rate);
14
15  /* Like read(), but restarts after EINTR, and reads until count bytes
16     are received or a timeout occurs. */
17  int saferead_timeout(int fd, void *buf, size_t count, int timeout_ms);
```

```
18
19 static inline int saferead(int fd, void *buf, size_t count) {
20         return saferead_timeout(fd, buf, count, -1);
21 }
22
23 /* Like write(), but restarts after EINTR */
24 ssize_t safewrite(int fd, const void *buf, size_t count);
25
26 /* Read bytes until no more are available for specified time */
27 int drain_timeout(int fd, int msec);
28
29 /* Read bytes until no more are available for 0.1 sec */
30 static inline int drain(int fd) {
31         return drain_timeout(fd, 100);
32 }
33
34 /* Like fprintf, but to a fd, using safewrite */
35 int fdprintf(int fd, const char *fmt, ...);
36
37 /* Like fgets, but from a fd, using saferead_timeout. */
38 char *fdgets(char *s, int size, int fd, int timeout_ms);
39
40 /* Like perl chomp. */
41 void chomp(char *s);
42
43 #endif
```

Listing G-34: `pc/serial-util.c`: Serial interface driver, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/serial-util.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
1  /*
2   * Serial I/O helper routines
3   *
4   * Jim Paris <jim@jtan.com>
5   * $Id$
6   */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <termio.h>
11 #include <sys/types.h>
12 #include <sys/ioctl.h>
13 #include <errno.h>
14 #include <unistd.h>
15 #include <string.h>
16 #include <fcntl.h>
17 #include <err.h>
18 #include <linux/serial.h>
19 #include <poll.h>
20 #include <stdarg.h>
21 #include "serial-util.h"
22
23 static int rate_to_constant(int baudrate) {
24 #define B(x) case x: return B##x
25         switch(baudrate) {
26                 B(50);     B(75);     B(110);    B(134);    B(150);
27                 B(200);    B(300);    B(600);    B(1200);   B(1800);
28                 B(2400);   B(4800);   B(9600);   B(19200);  B(38400);
29                 B(57600);  B(115200); B(230400); B(460800); B(500000);
30                 B(576000); B(921600); B(1000000);B(1152000);B(1500000);
31         default: return 0;
```

```
        }
#undef B
}

/* Open serial port in raw mode, with custom baudrate if necessary */
int serial_open(const char *device, int rate)
{
        struct termios options;
        struct serial_struct serinfo;
        int fd;
        int speed = 0;

        /* Open and configure serial port */
        if ((fd = open(device,O_RDWR|O_NOCTTY)) == -1)
                return -1;

        speed = rate_to_constant(rate);

        if (speed == 0) {
                serinfo.reserved_char[0] = 0;
                if (ioctl(fd, TIOCGSERIAL, &serinfo) < 0)
                        return -1;
                serinfo.flags &= ~ASYNC_SPD_MASK;
                serinfo.flags |= ASYNC_SPD_CUST;
                serinfo.custom_divisor = (serinfo.baud_base + (rate / 2)) / rate;
                if (serinfo.custom_divisor < 1)
                        serinfo.custom_divisor = 1;
                if (ioctl(fd, TIOCSSERIAL, &serinfo) < 0)
                        return -1;
                if (ioctl(fd, TIOCGSERIAL, &serinfo) < 0)
                        return -1;
                if (serinfo.custom_divisor * rate != serinfo.baud_base) {
                        warnx("actual baudrate is %d / %d = %f",
                              serinfo.baud_base, serinfo.custom_divisor,
                              (float)serinfo.baud_base / serinfo.custom_divisor);
                }
        }

        fcntl(fd, F_SETFL, 0);
        tcgetattr(fd, &options);
        cfsetispeed(&options, speed ?: B38400);
        cfsetospeed(&options, speed ?: B38400);
        cfmakeraw(&options);
        options.c_cflag |= (CLOCAL | CREAD);
        options.c_cflag &= ~CRTSCTS;
        if (tcsetattr(fd, TCSANOW, &options) != 0)
                return -1;

        return fd;
}

/* Like read(), but restarts after EINTR, and reads until count bytes
   are received or a timeout occurs. */
int saferead_timeout(int fd, void *buf, size_t count, int timeout_ms)
{
        struct pollfd pfd;
        int r;
        size_t nread = 0;

        while (count > 0) {
                pfd.fd = fd;
                pfd.events = POLLIN;
                r = poll(&pfd, 1, timeout_ms);
                if (r < 0 && errno == EINTR) /* retry */
                        continue;
                else if (r == 0) /* timeout */
                        return nread;
```

```
                else if (r == 1 && (pfd.revents & POLLIN)) { /* readable */
                        r = read(fd, buf, count);
                        if (r < 0 && errno == EINTR) /* retry */
                                continue;
                        if (r < 0) /* error */
                                return r;
                        if (r == 0) /* EOF */
                                return nread;
                        buf = (char *) buf + r;
                        count -= r;
                        nread += r;
                } else {
                        /* error */
                        return -1;
                }
        }
        return nread;
}

/* Like write(), but restarts after EINTR */
ssize_t safewrite(int fd, const void *buf, size_t count)
{
        size_t nwritten = 0;
        while (count > 0) {
                ssize_t r = write(fd, buf, count);

                if (r < 0 && errno == EINTR)
                        continue;
                if (r < 0)
                        return r;
                if (r == 0)
                        return nwritten;
                buf = (const char *)buf + r;
                count -= r;
                nwritten += r;
        }
        return nwritten;
}

/* Read bytes until no more are available for specified time */
int drain_timeout(int fd, int msec)
{
        char buf[1024];
        int ret;
        while (1) {
                ret = saferead_timeout(fd, buf, sizeof(buf), msec);
                if (ret <= 0)
                        return ret;
        }
}

/* Like fprintf, but to a fd, using safewrite. */
static int vfdprintf(int fd, const char *fmt, va_list args)
{
        static char buf[1024];
        vsnprintf(buf, sizeof(buf), fmt, args);
        return safewrite(fd, buf, strlen(buf));
}
int fdprintf(int fd, const char *fmt, ...)
{
        int ret;
        va_list args;
        va_start(args, fmt);
        ret = vfdprintf(fd, fmt, args);
        va_end(args);
        return ret;
}
```

```
166
167 /* Like fgets, but from a fd, using saferead_timeout. */
168 char *fdgets(char *s, int size, int fd, int timeout_ms)
169 {
170         int ret;
171         int nread = 0;
172
173         /* Not very efficient; needs to read one char at a time to
174          * avoid buffering */
175         while (nread < (size - 1)) {
176                 ret = saferead_timeout(fd, &s[nread], 1, timeout_ms);
177                 if (ret <= 0) {
178                         s[nread] = '\0';
179                         return NULL;
180                 }
181                 if (ret == 1) {
182                         nread++;
183
184                         /* found terminator? */
185                         if (s[nread-1] == '\n')
186                                 break;
187                 }
188         }
189         s[nread] = '\0';
190         return s;
191 }
192
193 /* Like perl chomp. */
194 void chomp(char *s)
195 {
196         int len = strlen(s);
197         /* do it twice to remove \r\n as well */
198         if (len > 1 && (s[len - 1] == '\r' || s[len - 1] == '\n'))
199                 s[--len] = '\0';
200         if (len > 1 && (s[len - 1] == '\r' || s[len - 1] == '\n'))
201                 s[--len] = '\0';
202 }
```

Listing G-35: `pc/gpib.h`: GPIB interface to Keithley test instruments, header file.

Git repository: https://git.jim.sh/jim/lees/zoom.git
Filename: pc/gpib.h
Revision: e28af0bb6747b0631f28ffd09813c45105e6906c

```
1  #ifndef GPIB_H
2  #define GPIB_H
3
4  int gpib_init(int fd);
5  int gpib_addr(int fd, int addr);
6
7  int keithley_init(int fd);
8  int keithley_current(int fd, double amps);
9  double keithley_read(int fd);
10 int keithley_off(int fd);
11
12 int keithley2002_init(int fd);
13 int keithley2002_init2(int fd);
14 int keithley2002_init_volts(int fd);
15 double keithley2002_read(int fd);
16 double keithley2002_read2(int fd, double* tx);
17 #endif
```

Listing G-36: `pc/gpib.c`: GPIB interface to Keithley test instruments, implementation.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pc/gpib.c`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

```
#include <stdio.h>
#include <string.h>
#ifndef __USE_ISOC99
#define __USE_ISOC99
#endif
#include <math.h>
#include "gpib.h"
#include "serial-util.h"

#define gputv(string, rv) do {      \
        int ____l = strlen(string); \
        if (safewrite(fd, string, ____l) != ____l) return rv; \
        if (safewrite(fd, "\r", 1) != 1) return rv; } while(0)
#define gput(string) gputv(string, -1)

int gpib_init(int fd)
{
        gput("++mode 1");
        gput("++auto 0");
        gput("++eos 1");
        gput("++eoi 1");
        gput("++read_tmo_ms 4000");
        gput("++eot_enable 1");
        gput("++eot_char 13");
        return 0;
}

int gpib_addr(int fd, int addr)
{
        char s[128];
        sprintf(s, "++addr %d", addr);
        gput(s);
        return 0;
}

int keithley_init(int fd)
{
        double i;
        gput(":syst:beep:stat 1");
        gput(":sour:func:mode curr");
        gput(":sour:del 0");
        gput(":sour:curr 0");
        gput(":sour:curr:range:auto on");
        gput(":outp on");
        drain(fd);
        i = keithley_read(fd);
        if (isnan(i))
                return -1;
        return 0;
}

int keithley_current(int fd, double amps)
{
        char s[128];
        sprintf(s, ":sour:curr %0.12f", amps);
        gput(s);
        return 0;
}

```

```
double keithley_read(int fd)
{
        char s[128];
        double i, dummy;
        gputv("read?", NAN);
        gputv("++read", NAN);
        if (fdgets(s, 128, fd, 2000) == NULL)
                return NAN;
        if (sscanf(s, "%lf,%lf,", &dummy, &i) != 2)
                return NAN;
        return i;
}

int keithley_off(int fd)
{
        gput(":outp off");
        return 0;
}


/* this function is for Keithley2002 Digital multimeter */
int keithley2002_init(int fd)
{
        double i;
        gput(":trac:cle");
        gput(":sens:curr:dc:rang:auto on");
        gput(":sens:curr:dc:dig 8");
        gput(":form:elem read");
        gput(":init:cont off");
        drain(fd);
        i = keithley2002_read(fd);
        if (isnan(i))
                return -1;
        return 0;
}

int keithley2002_init2(int fd)
{
        double i;
        gput(":trac:cle");
        gput(":sens:curr:dc:rang:auto on");
        gput(":sens:curr:dc:dig 8");
        gput(":form:elem read, time");
        gput(":syst:tst:type rel");
        gput(":syst:tst:rel:res");
        gput(":init:cont off");
        drain(fd);
        i = keithley2002_read(fd);
        if (isnan(i))
                return -1;
        return 0;
}

int keithley2002_init_volts(int fd)
{
        double i;
        gput(":trac:cle");
        gput(":sens:volt:dc:rang 5");
        gput(":sens:volt:dc:dig 8");
        gput(":form:elem read");
        gput(":init:cont off");
        drain(fd);
        i = keithley2002_read(fd);
        if (isnan(i))
                return -1;
        return 0;
}
```

```
127
128 double keithley2002_read(int fd)
129 {
130         char s[128];
131         double i;
132         gputv("read?", NAN);
133         gputv("++read", NAN);
134         if (fdgets(s, 128, fd, 2000) == NULL)
135                 return NAN;
136         if (sscanf(s, "%lf", &i) != 1)
137                 return NAN;
138         return i;
139
140 }
141
142 double keithley2002_read2(int fd, double *tx)
143 {
144         char s[128];
145         double i;
146         gputv("read?", NAN);
147         gputv("++read", NAN);
148         if (fdgets(s, 128, fd, 2000) == NULL)
149                 return NAN;
150         if (sscanf(s, "%lf,%lf", &i, tx) < 1)
151                 return NAN;
152         return i;
153
154 }
```

G.3 Schematics and PCB layout

Schematics and PCB layout for the Zoom NILM physically-windowed current sensor prototype, as described in Section 5.4, are shown in Figures G-1 to G-13. The analog portions of the design, shown in Figures G-3 and G-4, were developed in cooperation with Warit Wichakool [82].

Figure G-1: Control power schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

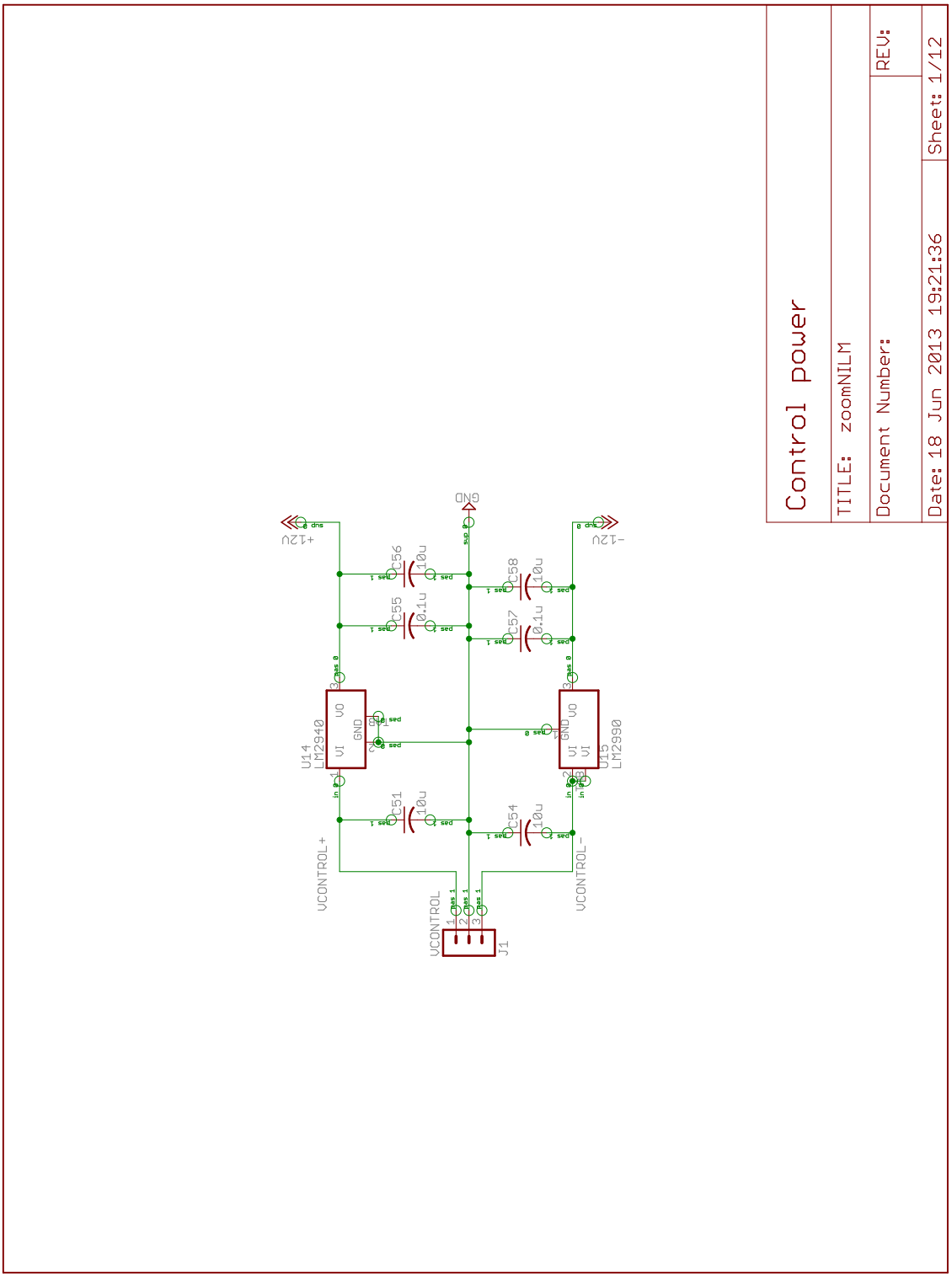


Figure G-2: Unregulated supply schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

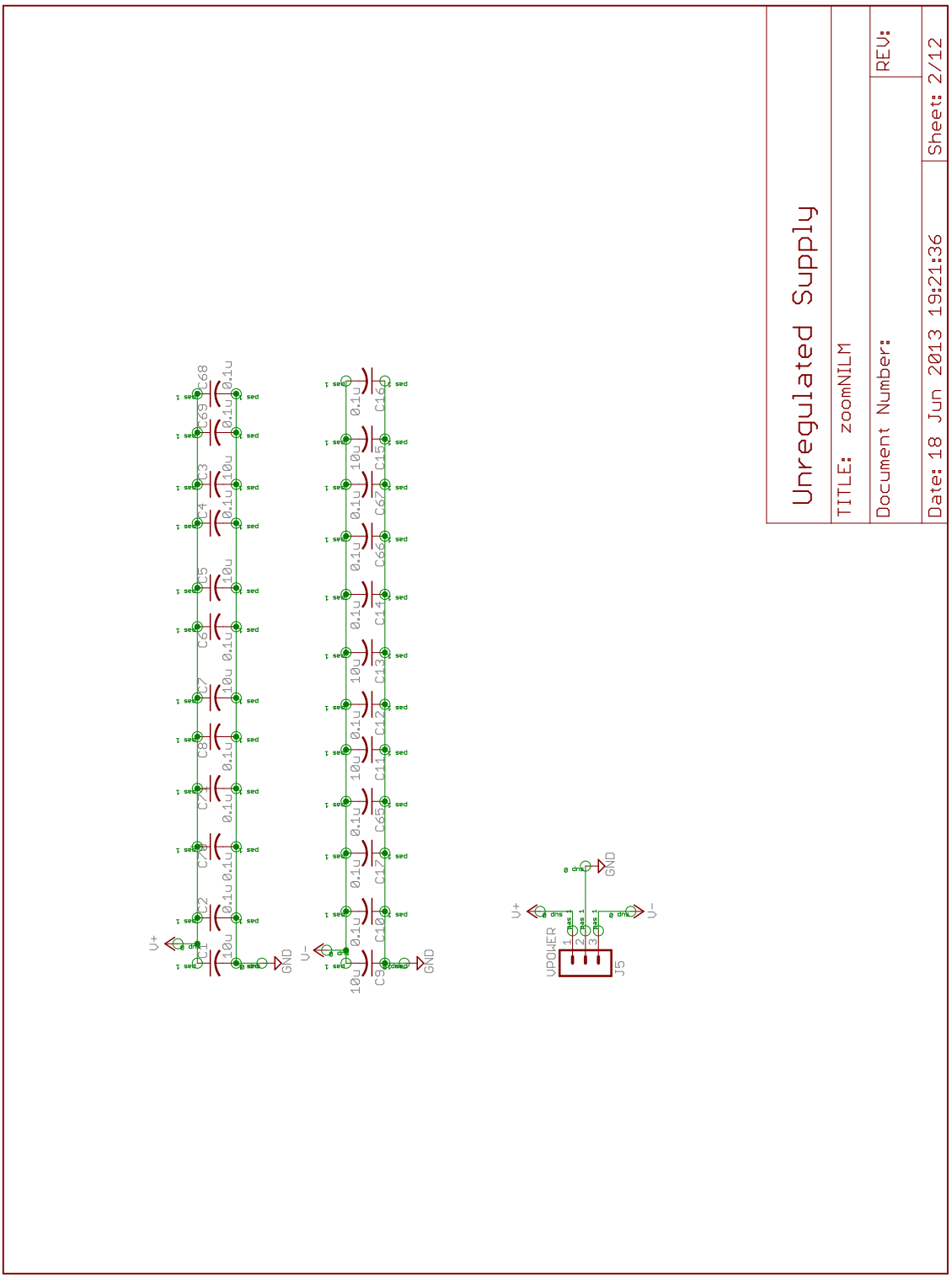


Figure G-3: OTA current source schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

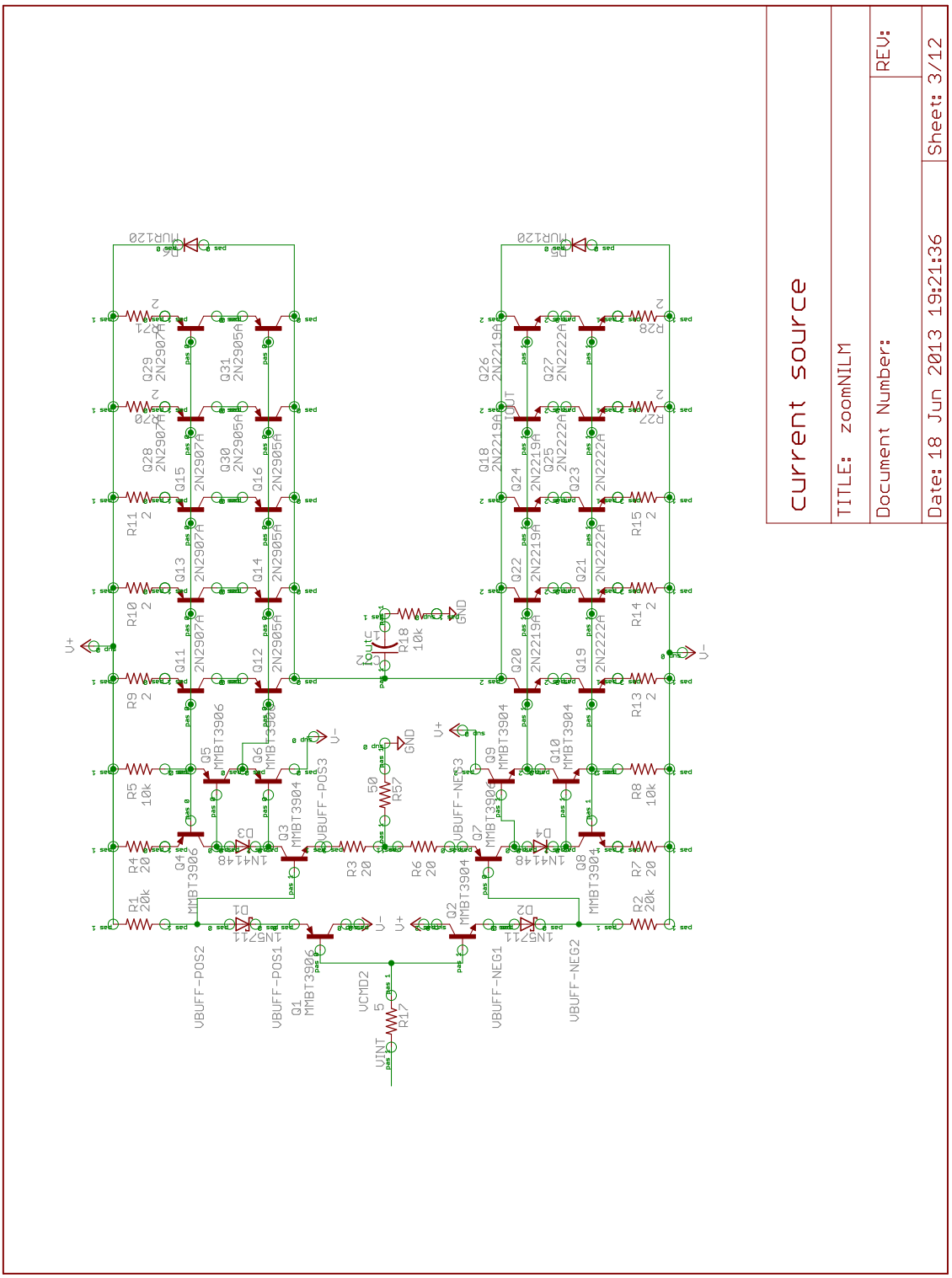


Figure G-4: Compensation current control circuit schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

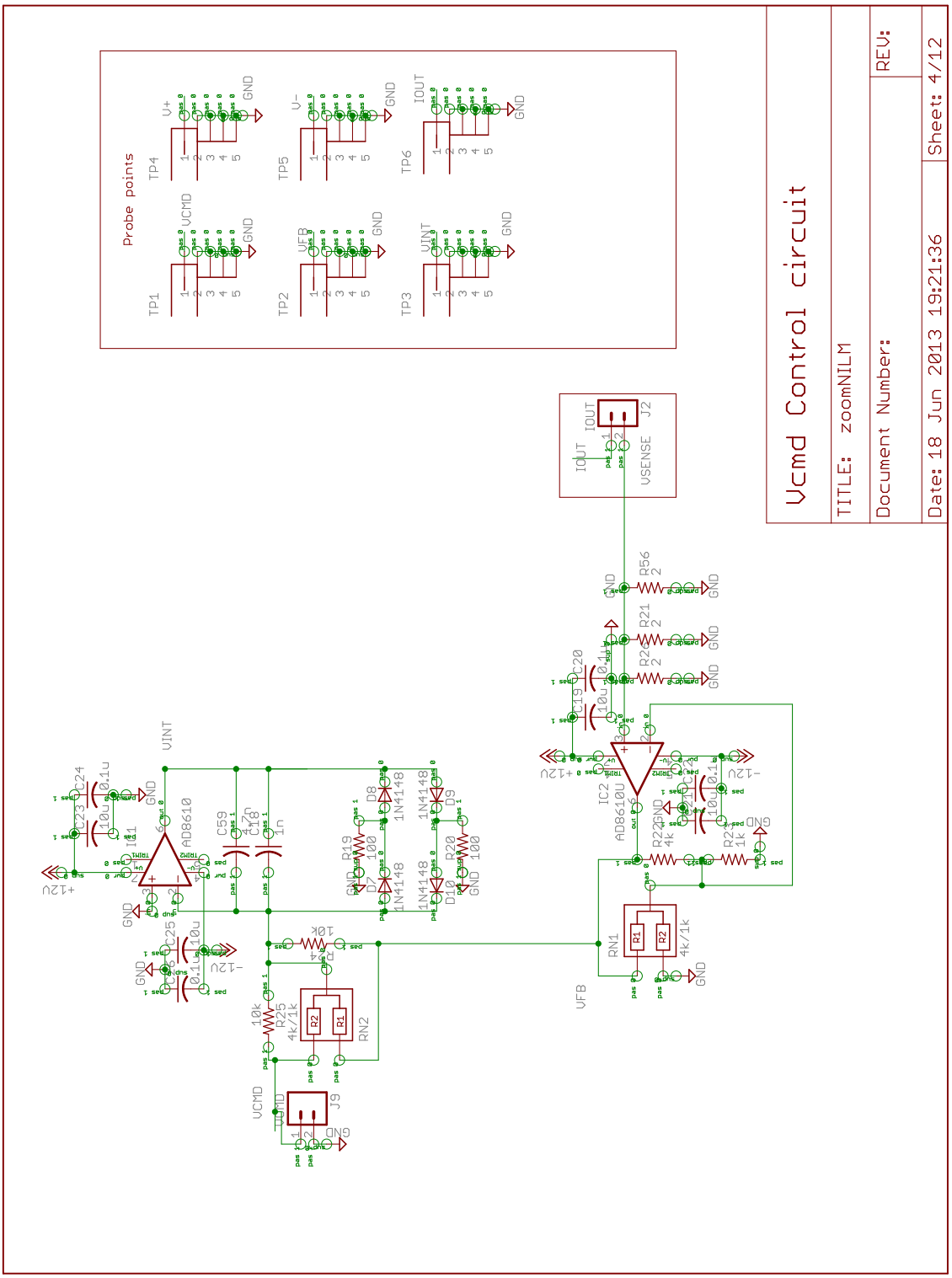


Figure G-5: DAC output stage schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

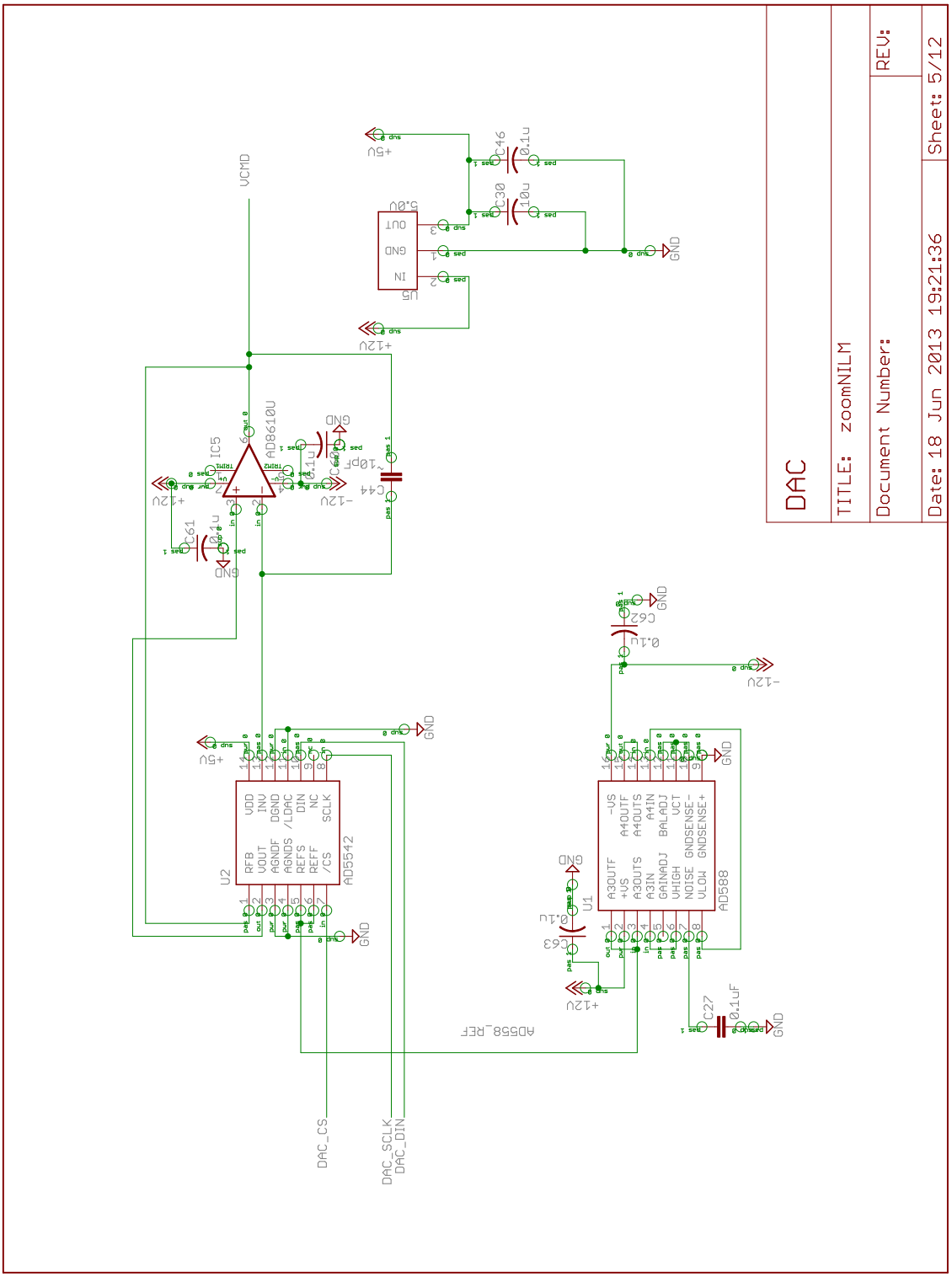


Figure G-6: PIC power supply and DAC optoisolation schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

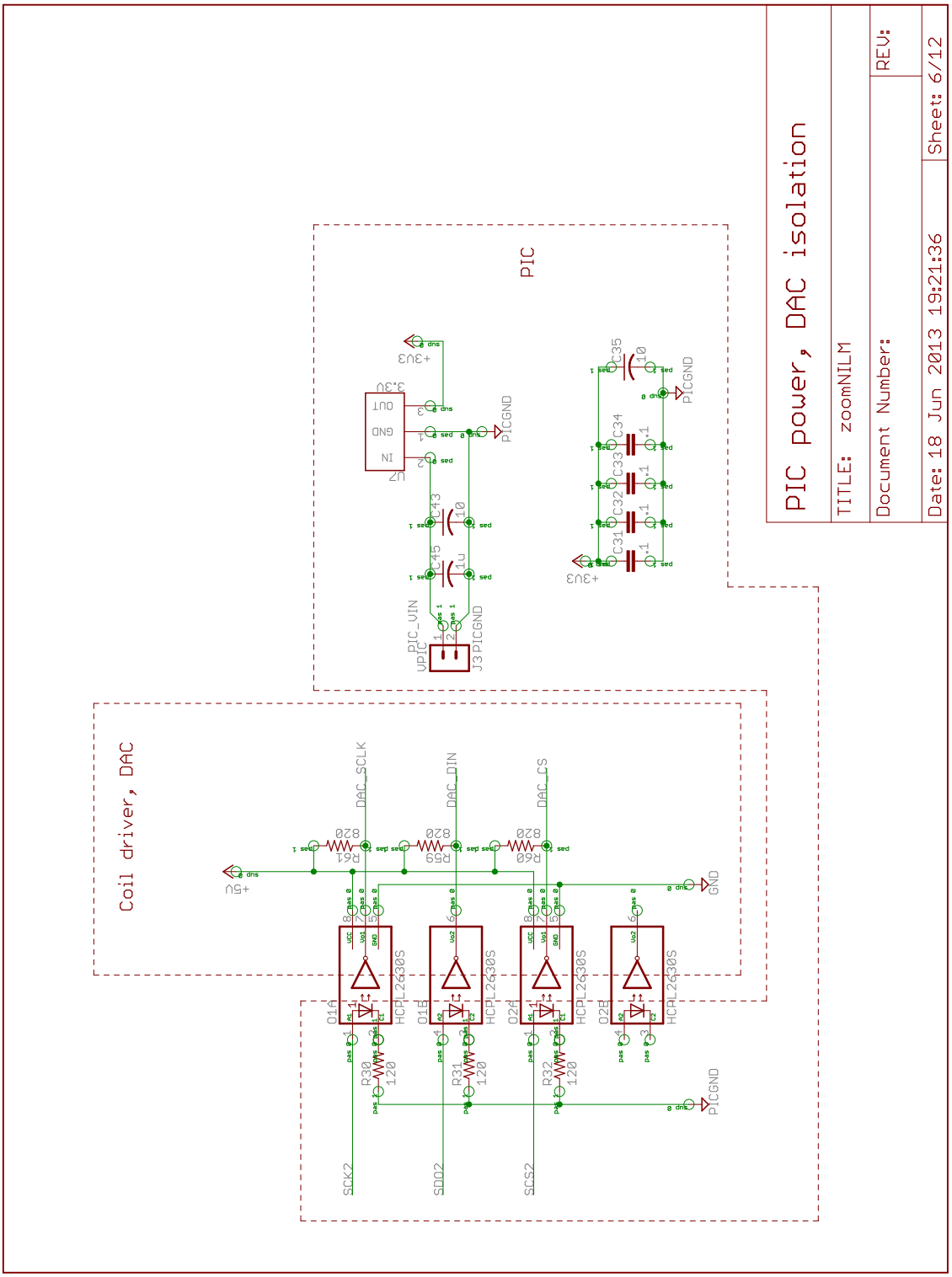


Figure G-7: USB communication interface schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

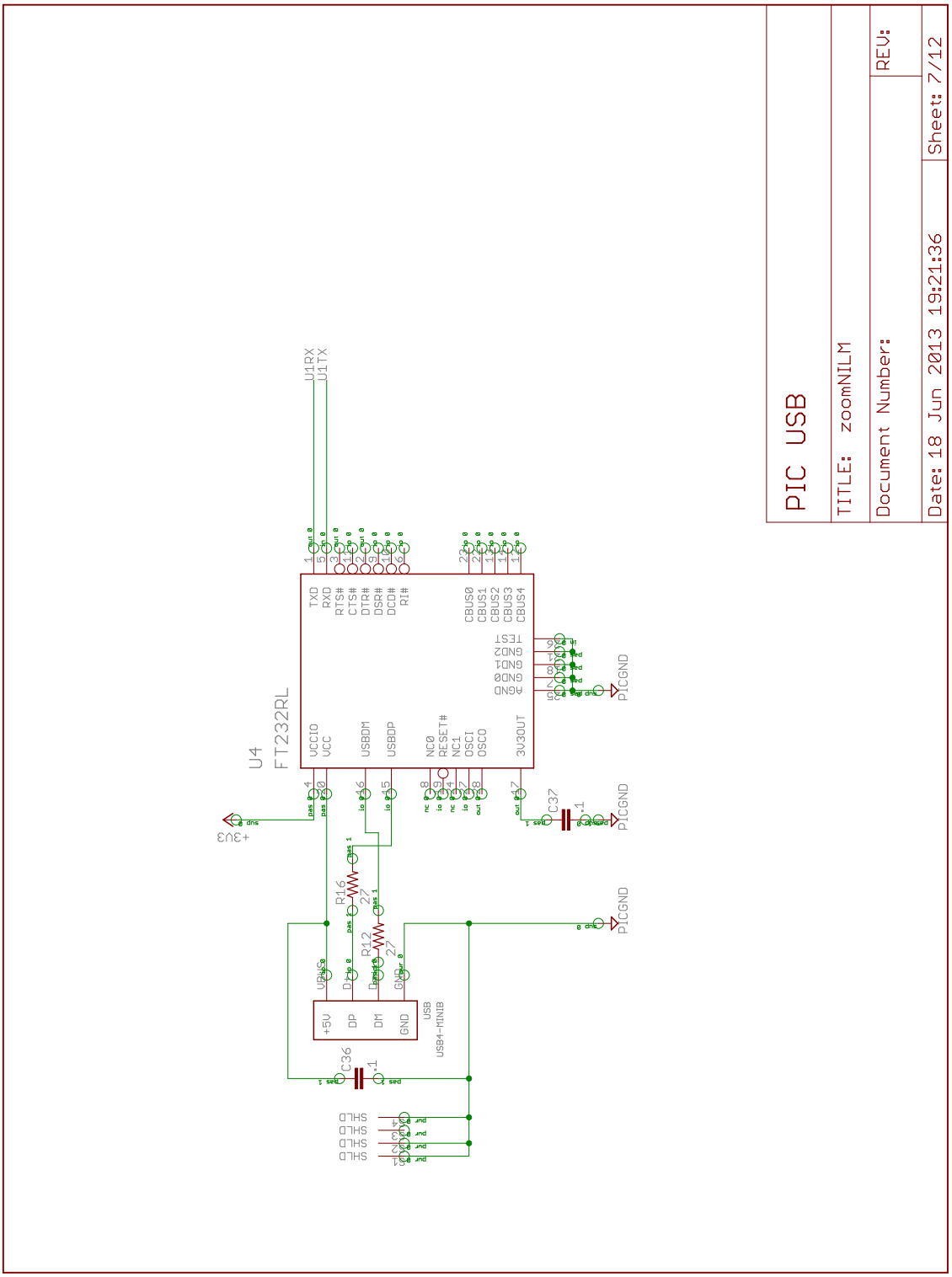


Figure G-8: Microchip dsPIC33FJ256GP710 PIC schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

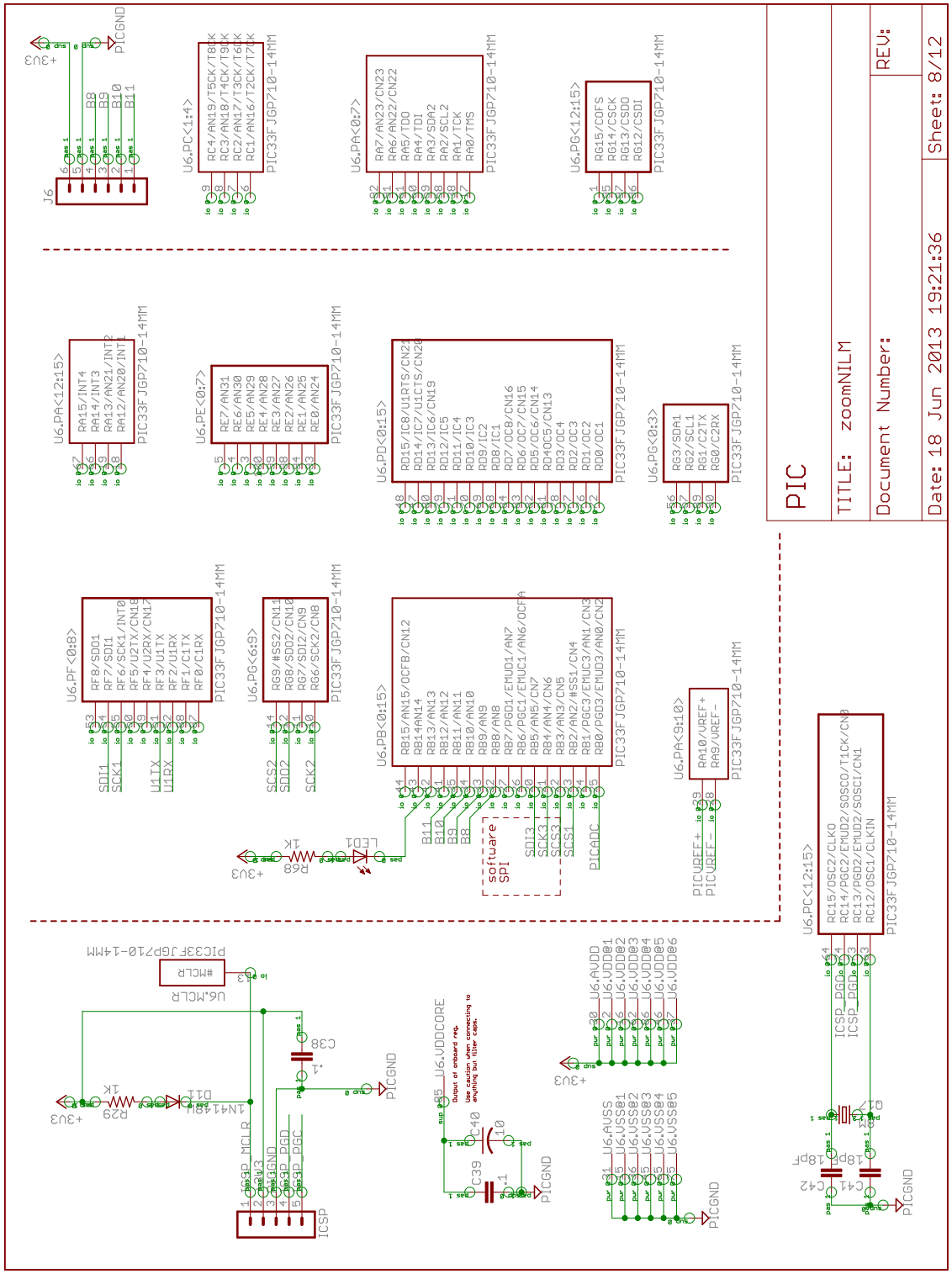


Figure G-9: Residual ADC schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

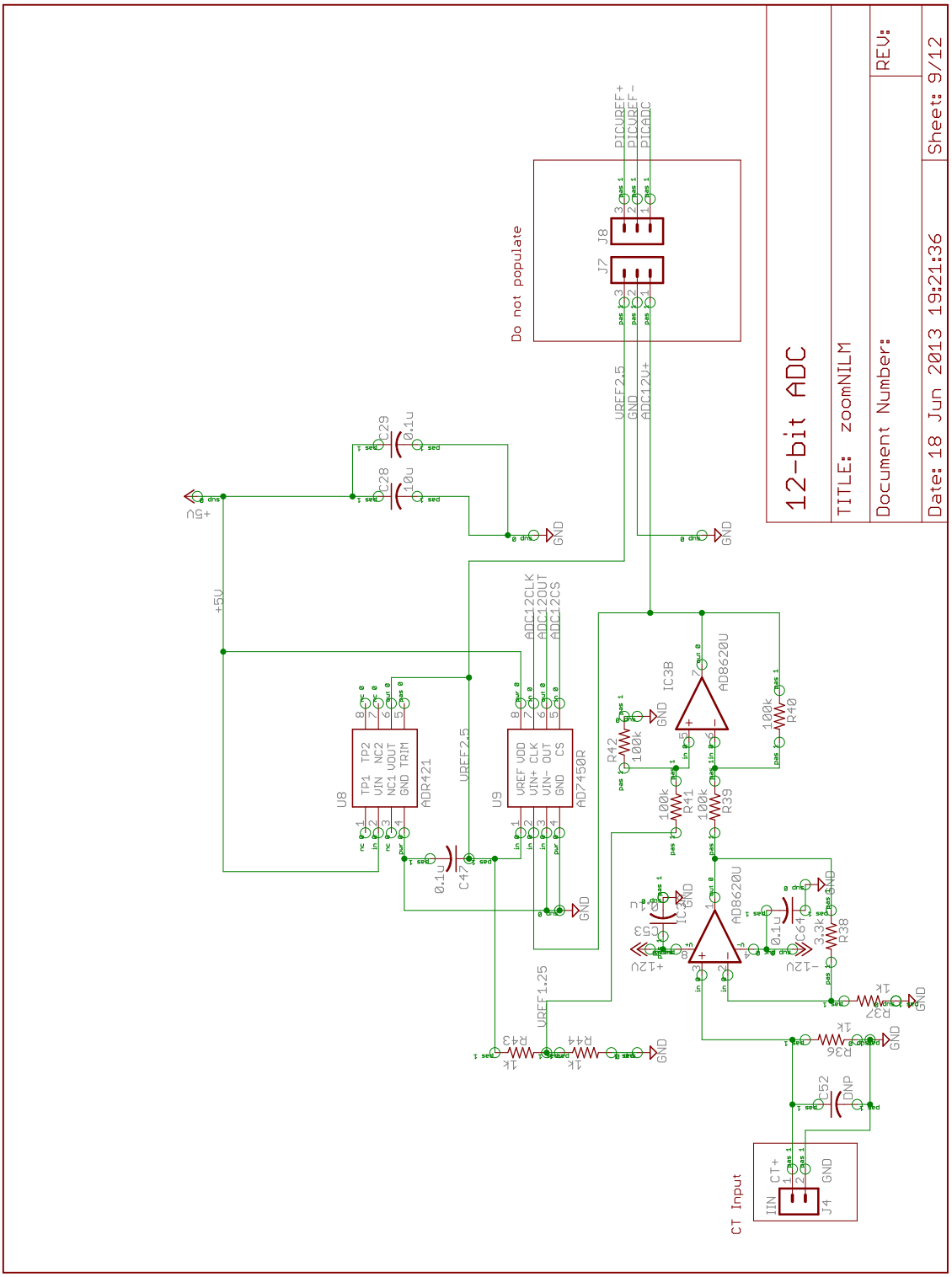


Figure G-10: Residual ADC optoisolation schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

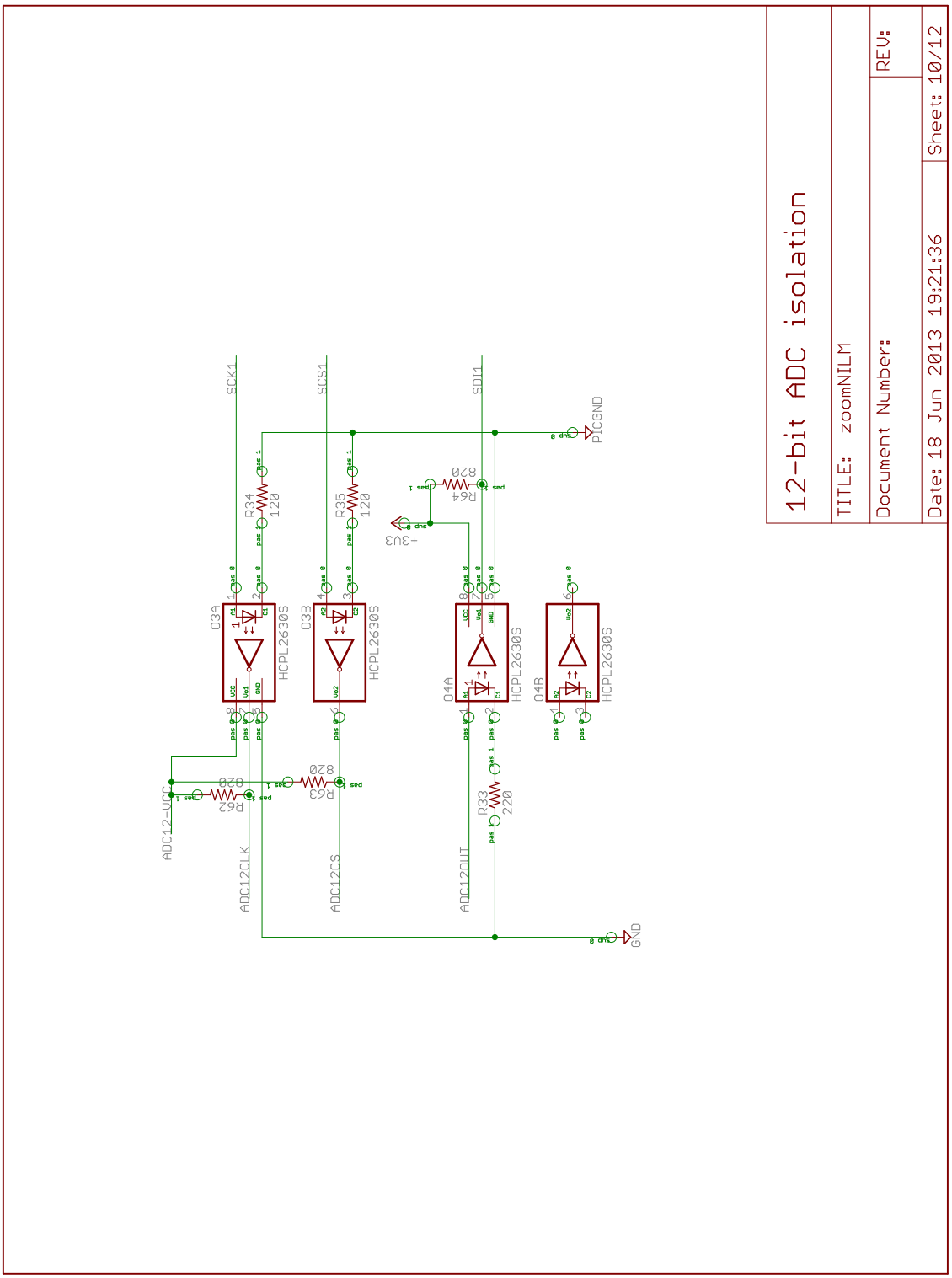


Figure G-11: Calibration ADC schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

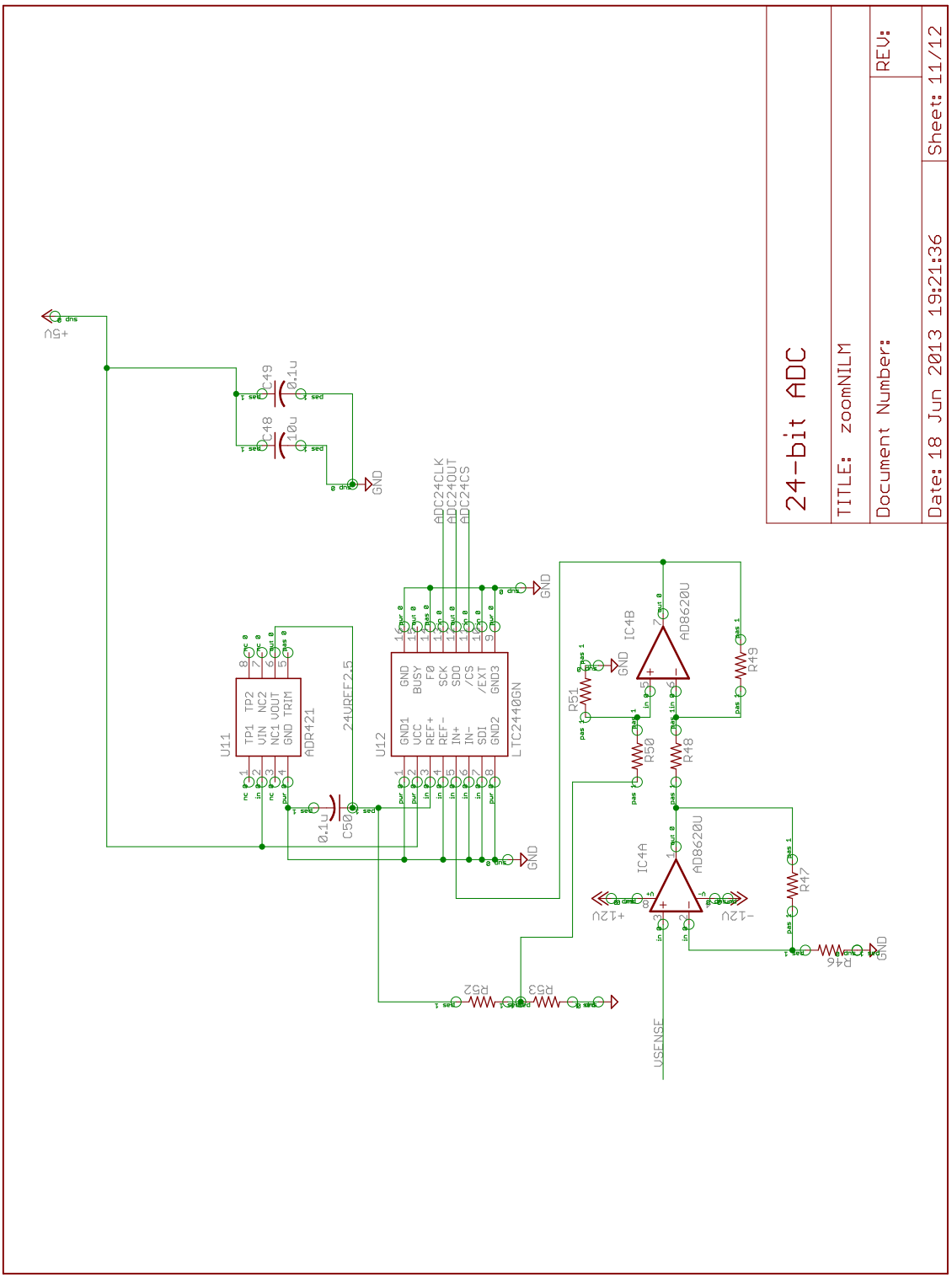


Figure G-12: Calibration ADC optoisolation schematic.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-schematic.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`

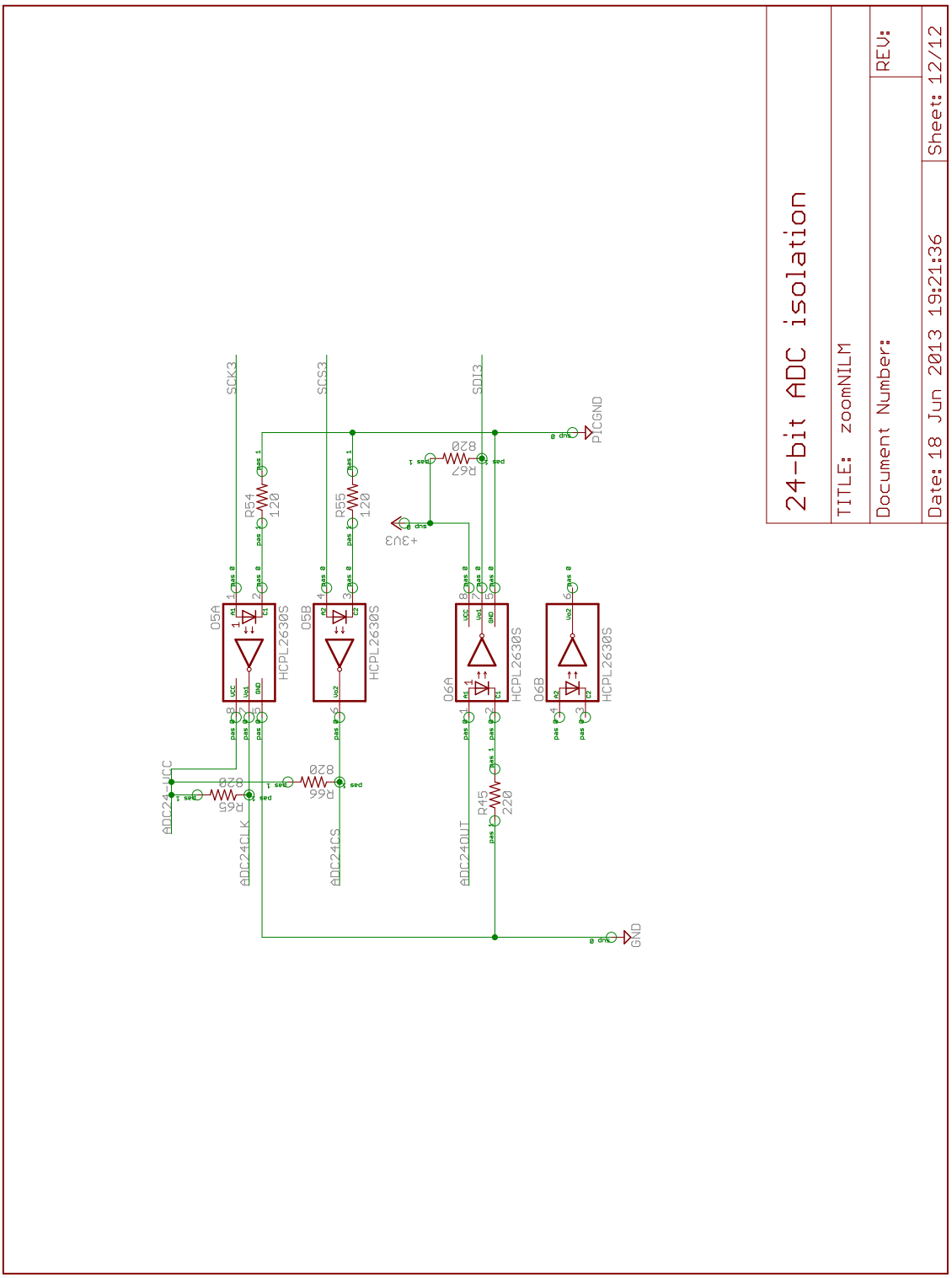
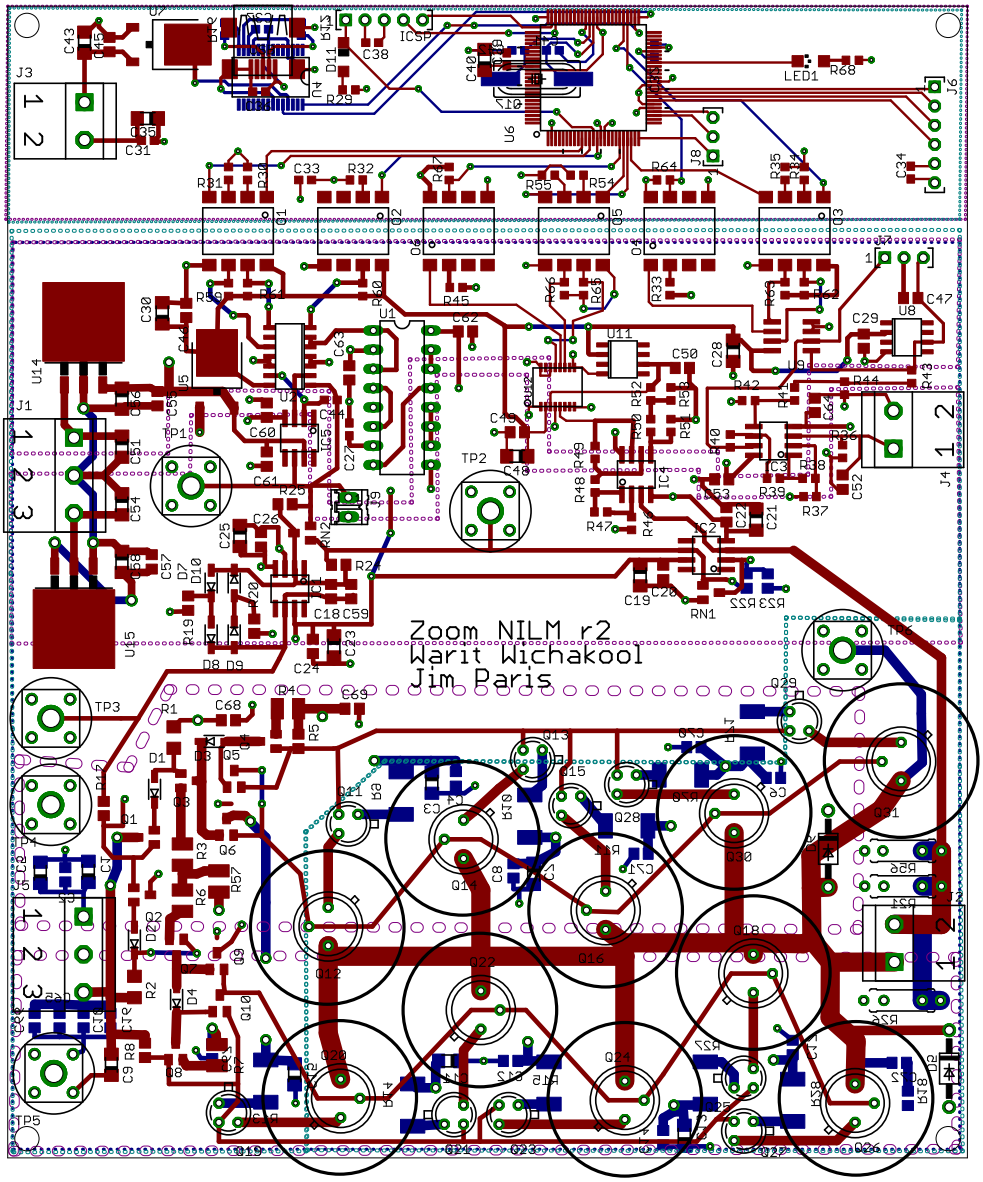


Figure G-13: Prototype PCB layout. The Zoom NILM prototype PCB is four layers and includes separate ground planes for the various analog and digital supplies. Microcontroller logic, USB communication, ADCs, DACs, and optoisolators are located in the upper half of the PCB.

Git repository: `https://git.jim.sh/jim/lees/zoom.git`
Filename: `pcb/zoom-r2/zoomNILM-pcb.pdf`
Revision: `e28af0bb6747b0631f28ffd09813c45105e6906c`



Bibliography

[1] S. B. Leeb, S. R. Shaw, and Jr. J. L. Kirtley. Transient event detection in spectral envelope estimates for nonintrusive load monitoring. *IEEE Transactions on Power Delivery*, 10(3):1200–1210, July 1995.

[2] L. K. Norford and S. B. Leeb. Non-intrusive electrical load monitoring in commercial buildings based on steady state and transient load-detection algorithms. *Energy and Buildings*, 24:51–64, 1996.

[3] U. A. Khan, S. B. Leeb, and M. C. Lee. A multiprocessor for transient event detection. *IEEE Transactions on Power Delivery*, 12(1):51–60, 1997.

[4] S. R. Shaw, S. B. Leeb, L. K. Norford, and R. W. Cox. Nonintrusive load monitoring and diagnostics in power systems. *IEEE Transactions on Instrumentation and Measurement*, 57(7):1445–1454, July 2008.

[5] G. R. Mitchell, R. W. Cox, J. Paris, and S. B. Leeb. Shipboard fluid system diagnostic indicators using non-intrusive load. *Naval Engineer's Journal*, 119(1), November 2007.

[6] W. Greene, J. S. Ramsey, S. B. Leeb, T. DeNucci, J. Paris, M. Obar, R. Cox, C. Laughman, and T. J. McCoy. Non-intrusive monitoring for condition-based maintenance. In *American Society of Naval Engineers Reconfigurability and Survivability Symposium*, Atlantic Beach, Florida, February 2005.

[7] T. DeNucci, R. Cox, S. B. Leeb, J. Paris, T. J. McCoy, C. Laughman, and W. Greene. Diagnostic indicators for shipboard systems using non-intrusive

load monitoring. In *IEEE Electric Ship Technologies Symposium*, Philadelphia, Pennsylvania, July 2005.

[8] J. P. Mosman, R. W. Cox, D. McKay, S. B. Leeb, and T. McCoy. Diagnostic indicators for shipboard cycling systems using non-intrusive load monitoring. In *American Society for Naval Engineers Day 2006*, Arlington, VA, June 2006.

[9] R. W. Cox, P. Bennett, D. McKay, J. Paris, and S. B. Leeb. Using the non-intrusive load monitor for shipboard supervisory control. In *IEEE Electric Ship Technologies Symposium*, Arlington, VA, May 2007.

[10] G. Mitchell, R. W. Cox, M. Piber, P. Bennett, J. Paris, W. Wichakool, and S. B. Leeb. Shipboard fluid system diagnostic indicators using nonintrusive load monitoring. In *American Society for Naval Engineers Day 2007*, Arlington, VA, June 2007.

[11] E. Proper, R. W. Cox, S. B. Leeb, K. Douglas, J. Paris, W. Wichakool, L. Foulks, R. Jones, P. Branch, A. Fuller, J. Leghorn, and G. Elkins. Field demonstration of a real-time non-intrusive monitoring system for condition-based maintenance. In *Electric Ship Design Symposium*, National Harbor, Maryland, February 2009.

[12] Z. Remscrim, J. Paris, S. B. Leeb, S. R. Shaw, S. Neuman, C. Schantz, S. Muller, and S. Page. FPGA-Based Spectral Envelope Preprocessor for Power Monitoring and Control. In *Applied Power Electronics Conference*, Palm Springs, CA, February 2010.

[13] J. Paris, Z. Remscrim, K. Douglas, S. B. Leeb, R. W. Cox, S. T. Gavin, S. G. Coe, J. R. Haag, and A. Goshorn. Scalability of non-intrusive load monitoring for shipboard applications. In *American Society of Naval Engineers Day 2009*, National Harbor, Maryland, April 2009.

[14] G. Mitchell. Shipboard fluid system diagnostics using non-intrusive load monitoring. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2007.

[15] P. Bennett. Using the non-intrusive load monitor for ship-board supervisory control. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2007.

[16] M. Piber. Improving shipboard maintenance practices using non-intrusive load monitoring. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2007.

[17] E. Proper. Automated classification of power signals. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2008.

[18] R. Jones. Improving shipboard applications of non-intrusive load monitoring. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2008.

[19] A. Jones. Harmonic approaches to non-intrusive load diagnostics. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2008.

[20] P. Branch. Development of real-time non-intrusive load monitor for shipboard fluid diagnostics. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2008.

[21] R. W. Cox. *Minimally Intrusive Strategies for Fault Detection and Energy Monitoring*. Phd, MIT, Department of Electrical Engineering and Computer Science, September 2006.

[22] J. Paris. A framework for non-intrusive load monitoring and diagnostics. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, February 2006.

[23] U. Orji, Z. Remscrim, C. Laughman, S. B. Leeb, W. Wichakool, C. Shantz, R. Cox, J. Paris, J. Kirtley, and L. Norford. Fault detection and diagnostics for

non-intrusive monitoring using motor harmonics. In *Applied Power Electronics Conference*, Palm Springs, CA, February 2010.

[24] T. DeNucci. Diagnostic indicators for shipboard systems using non-intrusive load monitoring. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2005.

[25] P. R. Armstrong. *Model Identification with Application to Building Control and Fault Detection*. Phd, MIT, Department of Architecture, September 2004.

[26] Christopher R Laughman, Peter R Armstrong, Leslie K Norford, and Steven B Leeb. The detection of liquid slugging phenomena in reciprocating compressors via power measurements. 2006.

[27] John Donnal, Uzoma Orji, Christopher Schantz, Jin Moon, Steven B Leeb, Jim Paris, Andrew Goshorn, Kevin Thomas, Jayme Dubinsky, and Robert Cox. Vampire: Accessing a life-blood of information for maintenance and damage assessment. *Proc. American Society of Naval Engineers*, 2012.

[28] D. J. Leeds. The Soft Grid 2013-2020: Big Data & Utility Analytics For Smart Grid. Technical report, GTM Research, Dec 2012.

[29] Microsoft Corporation. Microsoft Hohm Service Discontinuation. Available `http://web.archive.org/web/20120522051249/http://blog.microsoft-hohm.com/news/11-06-30/Microsoft_Hohm_Service_Discontinuation.aspx`. Accessed 2013-07-26.

[30] Google, Inc. An update on Google Health and Google PowerMeter. Available `http://googleblog.blogspot.com/2011/06/update-on-google-health-and-google.html`. Accessed 2013-07-26.

[31] CenterPoint Energy, Inc. Results of Pilot Project on Home Energy Use. Available `http://investors.centerpointenergy.com/releasedetail.cfm?ReleaseID=594825`. Accessed 2013-07-26.

[32] Fehrenbacher, Karen. 5 reasons Google PowerMeter didn't take off. Available http://web.archive.org/web/20130312025855/http://gigaom.com/2011/06/26/5-reasons-google-powermeter-didnt-take-off/#comment-634092. Accessed 2013-07-26.

[33] J. Donnal. Home NILM: A Comprehensive Non-Intrusive Load Monitoring Toolkit. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2013.

[34] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*. IEEE, New York, NY, USA, August 2008.

[35] D. Crockford. The application/json Media Type for Javascript Object Notation (JSON). RFC4627, IETF, Jul 2006.

[36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC2616, IETF, Jun 1999.

[37] CherryPy Team. CherryPy Documentation. Available http://docs.cherrypy.org/stable/. Accessed 2013-07-26.

[38] P. J. Eby. Python Web Server Gateway Interface v1.0.1. Python enhancement proposal, Sep 2010. Available http://www.python.org/dev/peps/pep-3333/.

[39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[40] Lance Williams. Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '83, pages 1–11, New York, NY, USA, 1983. ACM.

[41] E. W. Weisstein. Mathworld. Available http://mathworld.wolfram.com/. Accessed 2013-07-26.

[42] The Apache Software Foundation. Apache http server version 2.4 documentation. Available http://httpd.apache.org/docs/2.4/. Accessed 2013-07-26.

[43] The Apache Software Foundation. Apache http server authentication and authorization. Available http://httpd.apache.org/docs/current/howto/auth.html. Accessed 2013-07-26.

[44] T. D. McKay. Diagnostic indicators for shipboard mechanical systems using non-intrusive load monitoring. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, June 2006.

[45] Z. Clifford. An analog and digital data acquisition system for non-intrusive load monitoring. Masters thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2009.

[46] LabJack Corporation. Labjack ue9. Available http://labjack.com/ue9/. Accessed 2013-07-01.

[47] Anne van Kesteren. Cross-Origin Resource Sharing. Candidate recommendation, W3C, Jan 2013. Available http://www.w3.org/TR/cors.

[48] R. Fielding, L. Masinter, and T. Berners-Lee. Uniform Resource Identifier (URI): Generic Syntax. RFC3986, IETF, Jan 2005.

[49] ISO. *ISO/IEC 9899:2011 Information technology — Programming languages — C*. International Organization for Standardization, Geneva, Switzerland, December 2011.

[50] The Scipy Community. Structured arrays (aka "Record arrays"). Available http://docs.scipy.org/doc/numpy/user/basics.rec.html. Accessed 2013-07-26.

[51] Wikipedia. List of tz database time zones. Available http://en.wikipedia.org/wiki/List_of_tz_database_time_zones. Accessed 2013-08-04.

[52] P. Leach. A Universally Unique IDentifier (UUID) URN Namespace. RFC4122, IETF, Jul 2005.

[53] IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters. *IEEE Std 1241-2010 (Revision of IEEE Std 1241-2000)*, pages 1–139, 2011.

[54] S. R. Shaw and C. R. Laughman. A kalman-filter spectral envelope preprocessor. *IEEE Transactions on Instrumentation and Measurement*, 56(5):2010–2017, October 2007.

[55] S. R. Shaw. *System Identification Techniques and Modeling for Nonintrusive Load Diagnostics*. Phd, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, February 2000.

[56] C. R. Laughman, S. R. Shaw, S. B. Leeb, L. K Norford, R. W. Cox, K. D. Lee, and P. Armstrong. Power signature analysis. *IEEE Power and Energy Magazine*, pages 56–63, March 2003.

[57] S. B. Leeb. *A Conjoint Pattern Recognition Approach to Nonintrusive Load Monitoring*. Phd, MIT, Department of Electrical Engineering and Computer Science, February 1993.

[58] Hongbin Li, Petre Stoica, and Jian Li. Parameter estimation for harmonic sinusoidal signals. In *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, volume 2, pages 1047–1051 vol.2, 1999.

[59] A. Routray, A.K. Pradhan, and K.P. Rao. A novel kalman filter for frequency estimation of distorted signals in power systems. *Instrumentation and Measurement, IEEE Transactions on*, 51(3):469–479, 2002.

[60] W. M. Siebert. *Circuits, Signals and Systems*. The MIT Press, Cambridge, MA, 1986.

[61] W. Greene. Evaluation of non-intrusive monitoring for condition based maintenance applications on us navy propulsion plants. Master's thesis, Massachusetts

Institute of Technology, Department of Ocean Engineering and Department of Mechanical Engineering, June 2005.

[62] J. Paris. *A Comprehensive System for Non-Intrusive Load Monitoring and Diagnostics*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, September 2013.

[63] Jean-Marc Valin, Daniel V Smith, Christopher Montgomery, and Timothy B Terriberry. An iterative linearised solution to the sinusoidal parameter estimation problem. *Computers & Electrical Engineering*, 36(4):603–616, 2010.

[64] J. Schoukens, R. Pintelon, and H. Van Hamme. The interpolated fast fourier transform: a comparative study. *Instrumentation and Measurement, IEEE Transactions on*, 41(2):226–232, 1992.

[65] Tamás Zoltán Bilau, Tamás Megyeri, Attila Sárhegyi, János Márkus, and István Kollár. Four-parameter fitting of sine wave testing result: iteration and convergence. *Computer Standards and Interfaces*, pages 51–56, 2004.

[66] H. Renders, J. Schoukens, and G. Vilain. High-accuracy spectrum analysis of sampled discrete frequency signals by analytical leakage compensation. *Instrumentation and Measurement, IEEE Transactions on*, 33(4):287–292, 1984.

[67] R.M. Gray and Jr. Stockham, T.G. Dithered quantizers. *Information Theory, IEEE Transactions on*, 39(3):805–812, 1993.

[68] M.F. Wagdy and W.-M. Ng. Validity of uniform quantization error model for sinusoidal signals without and with dither. *Instrumentation and Measurement, IEEE Transactions on*, 38(3):718–722, 1989.

[69] L. Schuchman. Dither signals and their effect on quantization noise. *Communication Technology, IEEE Transactions on*, 12(4):162–165, 1964.

[70] K.D. Lee, S.B. Leeb, L.K. Norford, P.R. Armstrong, J. Holloway, and S.R. Shaw. Estimation of variable-speed-drive power consumption from harmonic content. *Energy Conversion, IEEE Transactions on*, 20(3):566–574, 2005.

[71] Steven B. Leeb, Steven R. Shaw, and James L. Kirtley. Transient event detection in spectral envelope estimates for nonintrusive load monitoring. *IEEE Trans. Power Del.*, 10(3):1200–1210, Jul 1995.

[72] J. S. Ramsey, S. B. Leeb, T. DeNucci, J. Paris, M. Obar, R. Cox, C. Laughman, and T. J. McCoy. Shipboard applications of non-intrusive load monitoring. In *American Society of Naval Engineers Reconfigurability and Survivability Symposium*, Atlantic Beach, Florida, February 2005.

[73] Derac Son and Johannes D. Seivert. A new current sensor based on the measurement of the apparent coercive field strength. *IEEE Trans. Instrum. Meas.*, 38(6):1080–1082, Dec 1989.

[74] Satoshi Ogasawara, Kazuhiro Murata, and Hirofumi Akagi. A digital current sensor for pwm inverters. In *Industry Applications Society Annual Meeting, 1992., Conference Record of the 1992 IEEE*, volume 1, pages 949–955, Oct 1992.

[75] Toshikatsu Sonoda, Ryuzo Ueda, and Kunio Koga. An ac and dc current sensor of high accuracy. *IEEE Trans. Ind. Appl.*, 28(5):1087–1094, Sept/Oct 1992.

[76] J. Pankau, D. Leggate, D. Schlegel, R. Kerkman, and G. Shibiniski. High frequency modeling of current sensors. *IEEE Trans. Ind. Appl.*, 35(6):1374–1382, Nov/Dec 1999.

[77] Dong Li and Guiyou Chen. A wide bandwidth current probe based on rogowski coil and hall sensor. In *Power Electronics and Motion Control Conference, 2006. IPEMC 2006. CES/IEEE 5th International*, volume 2, pages 1–5, Aug 2006.

[78] James Lenz and Alan S. Edelstein. Magnetic sensors and their applications. *IEEE Sensors J.*, 6(3):631–649, Jun 2006.

[79] Milan M. Ponjavić and Radivoje Durić. Nonlinear modeling of the self-oscillating fluxgate current sensor. *IEEE Sensors J.*, 7(11):1546–1553, Nov 2007.

[80] C. Moreland, F. Murden, M. Elliott, J. Young, M. Hensley, and R. Stop. A 14-bit 100-msample/s subranging adc. *IEEE J. Solid-State Circuits*, 35(12):1791–1798, Dec 2000.

[81] K. D. Hurst and T. G. Habetler. Sensorless speed measurement using current harmonic spectral estimation in induction machine drives. *IEEE Trans. Power Electron.*, 11(1):66–73, 1996.

[82] W. Wichakool. *Advanced Nonintrusive Load Monitoring System.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, February 2011.