Spinlab Slow Control Measurement System User Guide

Rev. 1.0

Steve Fromm*

(Dated: July 22, 2017)

Abstract

The Spinlab Slow Control (SpinlabSC) Measurement System encompasses all sensors, software systems and the database server that monitor and record all slow control variables in Spinlab (Room 1350/NSCL) and the experiments in the Radium-225 Electric Dipole Moment (RaEDM) and Single Atom Microscope (SAM) projects. All sensors are managed via a collection of LabVIEW Virtual Instruments (VI) and Python library modules, and record measurements to a MySQL database server. This user guide will provide an introduction to the system and demonstrate use of its various features. A full API reference is included at the end.

* fromm@nscl.msu.edu

CONTENTS

I. OVERVIEW

The SpinlabSC system is a combination of computers, data acquisition (DAQ) devices and software that operate with a shared data model. While every device will require custom built software, interactions with the database will be common across all devices. Similarly, access to the data for analysis will follow the same model, allowing a user to download and process data without the need to know the underlying SQL structure of the database. The following sections will provide an overview to the components involved in the system.

A. Data Model

The basis of the SpinlabSC system is its shared data model. The model represents both information about the involved devices and their measurement, and is structured in an object-oriented design. This allows for a user to create a program that interacts with tangible objects (sensors, devices, data sets, etc.) instead of dealing directly with the database structure (tables, rows, columns, etc.).

Individual sensors are categorized by a hierarchy of groupings, each represented by its own object. There are a predefined number of grouping caegories, which create a tree-like structure of all sensors that the SpinlabSC system manages. For example, a device object may be the parent object of multiple sensor objects. The following are the categories that all groups and sensors an belong to:

Owner: Owners represent the top level grouping of all sensor hierarchies. For instance, Spinlab would be the owner object of all sensors that belong to the lab, whereas items borrowed from another lab would belong to a separate owner object, such as ANL (Argonne National Laboratory).

Project: Projects represent the second level of the hierarchy. Each project will be a sub-grouping of a parent owner object. Projects should be logical distinctions of separate experiments and/or locations. Examples of projects would be RaEDM and General (pertaining to anything in room 1350 not specifically associated with a particular experiment).

System: Systems represent a standalone apparatus/category that belongs to a

parent object. AN example of a system would be the High Voltage Conditioning apparatus.

Device: Devices represent specific pieces of equipment that belong to a system. Devices are not directly responsible for making measurements, but will house and manage the sensors that do. An example of a device is a Digital Multimeter (DMM).

Sensor: Sensors are objects that are directly responsible for making a measurement that will be recorded to the database. Multiple sensors make belong to any one device. An example of sensors would be temperature and humidity probes that comprise a standalone temperature monitor.

Manufacturer: Manufacturers are objects that contain information about the manufacturer or distributer of a device. This allows for quick reference of detailed device data. Manufacturers are separate from the normal sensor grouping hierarchy.

The naming convention for any sensor or group comes directly from the model. Any group is named by listing all parent groups up to the objects own level. A full sensor name would then be formatted as:

```
Owner.Project.System.Device.Sensor
```

For example, the name of a temperature sensor monitoring the a location in the lab might be named:

```
Spinlab.General.Environment.TempRHS_1.Temperature
```

Measurements in the data model are represented as Record objects. These objects contain the data and timestamp of a measurement, and form RecordSet objects. There is one top level RecordSet to which all records belong, and from this structure record sets for a particular sensor and time ranges can be selected. All sensors record measurements to this top level data set.

An additional object in the data model is Units. Units define the units of measure that a sensor records its data in. Units will typically be predefined and shared across numerous sensors, and are used in the formatting, display and plotting of measurement data.

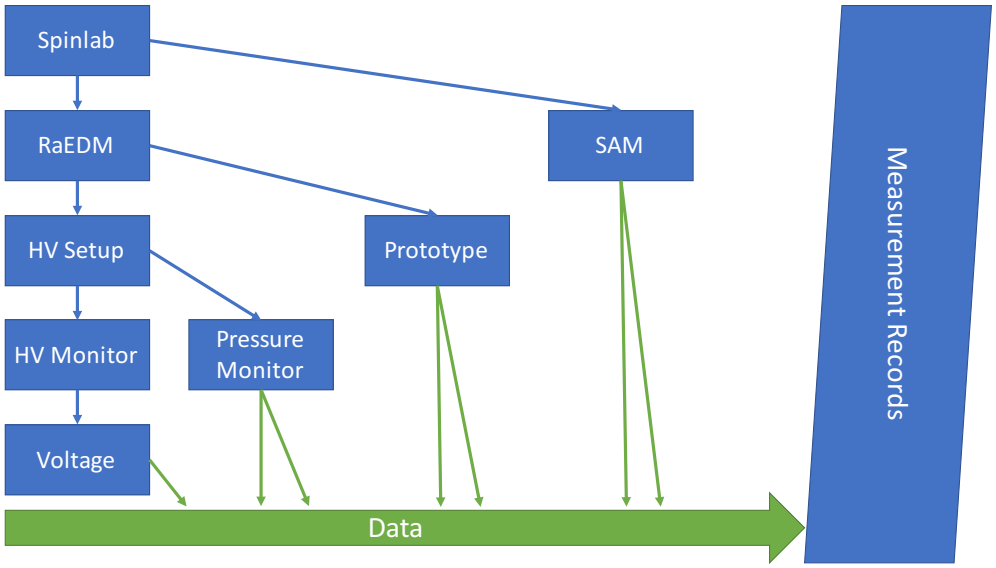

FIG. 1. Sample hierarchy of group levels.

B. Database Server

The database server is the main communication hub that all sensors and users interact and store data through. The physical machine runs a Debian Linux distribution and an Oracle MySQL server. The operating system and all binaries are stored on a 120 GB SSD, and two 1TB SATA drives house the data and back-ups respectively. The back-ups occur on an hourly basis, incrementally.

The MySQL server forms the core of the relational database used by the SpinlabSC system. It is accessible via the NSCL/FRIB intranet at:

URL: `spinlabdb.nscl.msu.edu`

Port: `3306`

A number of user accounts with varying privileges exist for sensors, software applications and end users. These user accounts and their associated password are available on a per device and application basis. New accounts can be generated via `root` login via SSH or using the MySQL Workbench software (downloadable from Oracle). At this time, there is no myPHPadmin installation associated with this database server.

The data model is implemented on the database server as a relational database created using SQL. All objects in the data model are represented in the database, and its structure has been normalized to 3NF to reduce any data redundancy. The database contains two identical

schema, `SpinlabSC`, which is the live version of the database, and `SpinlabSC_Playground` which is the development and debugging version of the database used for testing new features and software. See appendix for a full representation of the database.

C. Python Library

The entire data model and MySQL database connections are represented by the Python library module `SpinlabSC.py`. An objected-oriented design directly implements the objects in the data model, allowing users to directly use the group levels, sensors and associated objects in their programs. A separate database object encapsulates all of the underlying MySQL structure and procedures allowing the a user to deal solely with the objects in the data model and never write a line of SQL. A full reference of the classes and functions provided by the library are included in the appendices.

D. LabVIEW Virtual Instruments

Similar to the Python library, a collection of LabVIEW VIs encapsulates the data model model and provides an interface for communicating with the database server. These VIs perform one time tasks, such as selecting sensor information, writing a measurement to the database, etc., allowing them to be added in to existing VIs as sub-VIs within the main program loop. They were designed to take minimal execution time, so as to not significantly interfere with any existing timing considerations built into the the main VI. See the appendix for a full reference of the LabVIEW VIs available.

II. PROGRAMMING WITH THE SPINLABSC PYTHON LIBRARY

The SpinlabSC Python library module provides a full interface to the database that precludes the use of SQL. All common tasks that can be performed on the database are fully replicated in a series of Python classes and functions. The following sections will demonstrate how to perform these tasks in context of creating and using a new sensor.

A. Setting Up SpinlabSC Python Library

Before being able to utilize the SpinlabSC Python library, a few steps must be taken set up the library for use with your application or device:

1. Move a copy of the library file, `SpinlabSc.py` located in

   ```
   I:\projects\spinlab\software\Slow Control
   ```

   to the same directory as the Python program that you are using it in.

2. The underlying MySQL interface depends on the MySQL Connector for Python package from Oracle. The easiest way to install this library is from the IPython terminal using `pip`. To avoid any errors, it is best to use the URL for the download in the `pip` command (shown in listing 1).

   Listing 1. Installing MySQL Connector

   ```
   1 !pip install --egg http://dev.mysql.com/get/Downloads/
       Connector-Python/mysql-connector-python-2.1.6.zip
   ```

   Note: SpinlabSC.py has been tested with MySQL Connector for Python version 2.1.6. Newer versions may work, but version 2.1.6 is recommended for compatibility.

3. To verify that the library and its dependancies are installed correctly, open a Python terminal in the same directory that the copy of `SpinlabSC.py` was placed in, and import it (shown in listing 2). If everything is installed correctly, the import will be successful and no error messages will be displayed.

Listing 2. Importing the SpinlabSC library

```
1 import SpinlabSC
```

B. Establishing a Connection to the Database Server

Before being able to interact with the database, a connection to the server must be established. This process is shown in listing 3. Note that an actual username and password are not displayed, and will be provided on an as-needed basis.

Listing 3. Connecting to the database server

```
1  import SpinlabSC as ssc
2
3  # Configure the connection parameters
4  host = "spinlabdb.nscl.msu.edu"
5  dbname = "SpinlabSC"
6  username = "spinlab_app"
7  password = "********"
8
9  # Connect to the database server and store the connection in a
       Database object
10 db = ssc.Database(host, dbname, username, password)
```

C. Adding a New Sensor to the Database

Before being able to record measurements to the database, a sensor, and its grouping heirarchy, must be configured in the database. The following sections demonstrate how to add each level of groupings needed to use a sensor. Note that these grouping levels only need to be created once initially. Subsequent sensors added to the same owner/project/system/device only need to reference the existing grouping in the database.

1. Creating a New Owner

Owners represent the particular lab or institution that any sensor belongs to. The example in listing 4 shows how the `Spinlab` owner group could be created in the database.

Listing 4. Creating a new owner in the database

```
1  import SpinlabSC as ssc
2
3  # Connect to the database (parameters configured in a previous
       example)
4  db = ssc.Database(host, dbname, username, password)
5
6  # Configure parameters for this owner
7  name = "Spinlab"
8  description = "Jaideep Singh\'s Research Group"
9
10 # Add this owner to the database (returns reference to the
       created object)
11 owner = db.CreateNewOwner(name, description)
```

2. Creating a New Project

Projects represent the experiment or location that a sensor is part of. The example in listing 5 shows how the `Spinlab.General` project, which encompasses everything in the lab space that does not belong to a particular experiment, could be created.

Listing 5. Creating a new project in the database

```
1  import SpinlabSC as ssc
2
3  # Connect to the database (parameters configured in a previous
       example)
4  db = ssc.Database(host, dbname, username, password)
5
```

```
# Configure parameters for this project
name = "Spinlab.General"
description = "Anything within the lab space (room 1350) that does not belong to a particular experiment"

# Add this project to the database (returns reference to the created object)
project = db.CreateNewProject(name, description)
```

3. Creating a New System

Systems represent the apparatus or classification that a sensor belongs to. The example in listing 6 shows how the `Spinlab.General.Environment` system, which includes all environmental monitors in the lab space, could be created.

Listing 6. Creating a new system in the database

```
import SpinlabSC as ssc

# Connect to the database (parameters configured in a previous example)
db = ssc.Database(host, dbname, username, password)

# Configure parameters for this system
name = "Spinlab.General.Environment"
description = "Environmental monitors in the the lab (room 1350)"

# Add this system to the database (returns reference to the created object)
system = db.CreateNewSystem(name, description)
```

4. Creating a New Device Manufacturer

Manufacturers represent the company that produced a particular device, or the vendor it was obtained from. Manufacturers exist in the database to provide quick access to information about the device. The example in listing 7 shows how to set up a manufacturer in the database.

Listing 7. Creating a new manufacturer in the database

```
1  import SpinlabSC as ssc
2
3  # Connect to the database (parameters configured in a previous
       example)
4  db = ssc.Database(host, dbname, username, password)
5
6  # Configure parameters for this manufacturer
7  name = "Some Random Company"
8  description = "Produces random devices."
9  URL = "http://some-web-site"
10
11 # Add this manufacturer to the database (returns reference to
       the created object)
12 mfg = db.CreateNewManufacturer(name, description, URL)
```

5. Creating a New Device

Devices represent a physical entity that contains one or more sensors. An example of this would be a data logger that measures both temperature and humidity. The example in listing 8 shows how to set up a similar device in the database.

Listing 8. Creating a new device in the database

```
1  import SpinlabSC as ssc
2
```

```
# Connect to the database (parameters configured in a previous example)
db = ssc.Database(host, dbname, username, password)

# Configure parameters for this device
name = "Spinlab.General.Environment.TempRHS_1"
description = "Temperature and humidity data logger located near lab entrance"
URL = "http://location-of-device-documentation-on-wiki"

# Obtain reference to the device's manufacturer
mfg = db.GetManufacturer("Some Random Company")

# Add this device to the database (returns reference to the created object)
device = db.CreateNewDevice(name, description, URL, mfg)
```

6. Creating a New Unit of Measure

Units represent the units of measure associated with the measurements that a sensor records. Unit objects provide an easy way to access and display information about the measurement units. An example of a unit object for temperature being added to the database is shown in listing 9.

Listing 9. Creating new units of measure in the database

```
import SpinlabSC as ssc

# Connect to the database (parameters configured in a previous example)
db = ssc.Database(host, dbname, username, password)

# Configure parameters for these units
```

```
shortName = "deg C"
longName = "degrees Celsius"
description = "Temperature"

# Add these units to the database (returns reference to created
    object)
units = db.CreateNewUnits(shortName, longName, description)
```

7. Creating a New Sensor

Sensors represent the probes and detectors that record measurements. An example of this would be the temperature sensor in a data logger. The example in listing 10 shows how to set up this sensor in the database.

Listing 10. Creating a new sensor in the database

```
import SpinlabSC as ssc

# Connect to the database (parameters configured in a previous
   example)
db = ssc.Database(host, dbname, username, password)

# Configure parameters for this device
name = "Spinlab.General.Environment.TempRHS_1.Temperature"
description = "Measures temperature in degrees Celsius"

# Obtain reference to the device's units
units = db.GetUnits("degrees Celsius")

# Add this sensor to the database (returns reference to the
   created object)
device = db.CreateNewSensor(name, description, units)
```

D. Recording a Measurement to the Database

The SpinlabSC Python library provides the functionality to write a measurement to the database. The sensor, or a program controlling the sensor, are responsible for actually taking the measurement. The value of the measurement, and any uncertainty, can then be passed to the database. The example in listing 11 demonstrates how to configure a simple repeating timed measurement for the previously created temperature sensor.

Listing 11. Recording a measurement to the database

```
1  import SpinlabSC as ssc
2  import time
3
4  # Connect to the database (parameters configured in a previous
       example)
5  db = ssc.Database(host, dbname, username, password)
6
7  # Obtain the sensor object for these measurements
8  sensor = db.GetSensor("Spinlab.General.Environment.TempRHS_1.
       Temperature")
9
10 # Set up a loop for an hour worth of measurements
11 for i in range(60):
12         # Take a measurement
13         y = FunctionToTakeMeasurement()
14         dy = FunctionToCalculateUncertainty()
15
16         # Write this to the database (returns object of the
               stored record)
17         record = db.RecordMeasurement(sensor, y, dy)
18
19         # Wait until a minute has passed
20         time.sleep(60)
```

By default, the database will assign the timestamp of the insertion of the measurement to the record. A different timestamp can be passed in to the `RecordMeasurement()` function if the measurement occurred at an earlier time.

E. Reading Data from the Database

Data is accessed via RecordSet objects. These objects contain a set of measurement records for a particular sensor in a given timestamp range. The example in listing 12 shows how to obtain a RecordSet from the database as well as how to plot and save this data to a file.

Listing 12. Reading a record set from the database

```
1  import SpinlabSC as ssc
2  from datetime import *
3  from dateutils.relativedelta import *
4  import matplotlib.pyplot as plt
5
6  # Connect to the database (parameters configured in a previous
       example)
7  db = ssc.Database(host, dbname, username, password)
8
9  # Obtain the sensor object for these measurements
10 sensor = db.GetSensor("Spinlab.General.Environment.TempRHS_1.
       Temperature")
11
12 # Build a one week date range
13 endTime = datetime.now()
14 startTime = endTime + relativedelta(weeks = -1)
15
16 # Pull these records from the database
17 records = db.GetRecords(sensor, startTime, endTime)
18
19 # Grab x, y, dy values
```

```
20 x = records.times
21 y = records.data
22 dy = records.error
23
24 # Use matplotlib to plot this data
25 plt.errorbar(x, y, yerr=dy)
26 plt.xlabel("Timestamp")
27 plt.ylabel(records.GetUnitsLabel())
28 plt.title(records.GetPlotLabel())
29 plt.show()
30
31 # Finally, save this data as a CSV file, tab-delimited, with
      column headers
32 records.WriteCSV("data.csv", delim="\t", headers=True)
```

III. USING THE SPINLABSC LABVIEW VIRTUAL INSTRUMENTS

The SpinlabSC LabVIEW VI interface provides a way to add database communications into existing VI's. The VI's are built to open a connection, write/read data once and then close the connection before allowing the main calling VI to continue execution.

All of SpinlabSC LabVIEW VI's are located on the share drive at:

`I:\projects\spinlab\vis_Labview\Database VIs\`

A. Using Record-Measurement.vi

Record-Measurement.vi is the easiest way to write to the SpinlabSC database. This VI can be ran independent as a sub VI outside of the main program loop, and pass data points to the timed measurement setup via local variables. An example use of this VI is shown in figure 2.

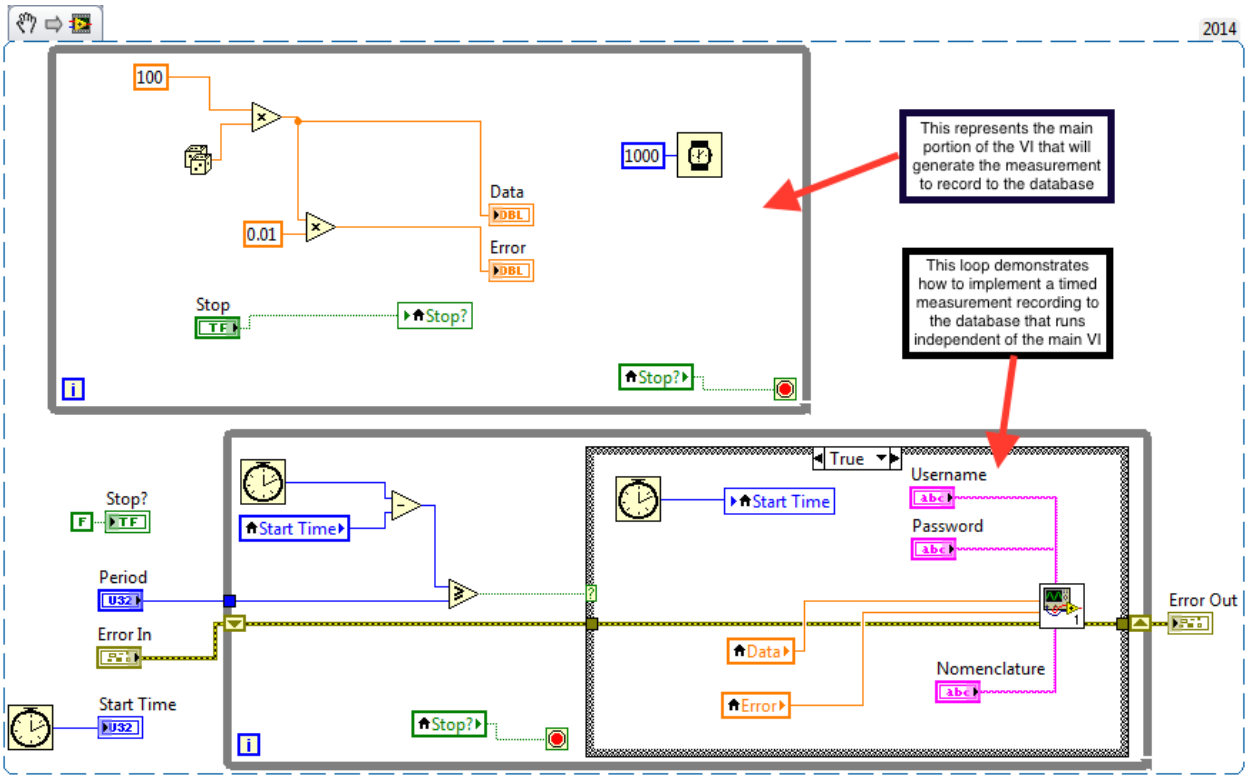

FIG. 2. This is an example of how to periodically record a measurement to the database without interfering with main program execution.

This VI has eight input parameters and one output:

Username: string, username of the database account

Password: string, hidden, password associated with the provided username

Table: string, name of the records table; allows access to devmode tables for testing

Nomenclature: string, full name of the sensor, in the format

`Owner.Project.System.Device.Sensor`

Data: double, value of the measurement

Error: double, uncertainty in the measurement

Error In/Error Out: LabVIEW error controls, indicate any errors and provides program flow control

B. Miscellaneous Sub-VI's

There are also a number of sub-VI's that handle specific portions of the measurement recording process. A series of string manipulations and checks against the database translate the nomenclature string into a numerical ID to reference in the database. If any part of the nomenclature hierarchy is not present in the respective database table, the VI will generate a `Sensor not found` error with an error code of `-1`. Any capitalization in the nomenclature string is ignored during the look-up process. All database communications are handled internally using the LabVIEW Database Connectivity Toolkit.

The following are brief description of the component sub-VI's:

Split-String.vi: This VI will take an input string and a token to split the string around. While used for extracting each group level name from the nomenclature string, this VI is generic enough to also parse CSV, tab-delimited and similar data formats.

Get-Sensor.vi: This VI takes an input nomenclature string and returns the associated sensor ID, or will produce an error if the sensor does not exist in the database.

Record-Measurement.vi: This is the main VI implementing connectivity with the database. Use this VI in existing programs to write a measurement to the database.

Time Measurements Example.vi: This VI demonstrates a way to integrate a timed loop responsible for recording measurements that runs in parallel to an existing program execution loop. This allows for simplified integration into an existing program. Local variables of existing data point indicators can pass information to the Record-Measurement.vi, and a boolean toggle can be tied to an existing stop button to terminate the measurement recording loop.

Write-to-SpinlabDB.vi: This VI is similar to Record-Measurement.vi, but requires the sensor ID to be input directly. Optionally, a table name can be input, allowing for easier testing of software using devmode tables.

C. Connecting to the Database on Windows

For any LabVIEW VI running on a Windows computer, one additional file, `Spinlab DB.udl` is required. Windows requires this file to configure the ODBC data source and connect to the database server. The server connection is established remotely, so this file is not tied to any one physical machine.

One additional consideration must be made to use the underlying Database Connectivity Toolkit on Windows. This LabVIEW package is only available in the 32-bit version of the program, so any 64-bit VI using the database VI's must be manually opened in 32-bit mode.

The Database Connectivity Toolkit requires the MySQL Connector/ODBC divers for Windows. To make sure that LabVIEW and Windows can communicate with the database, you will need to download and install BOTH the 32-bit and the 64-bit drivers located at:

`https://dev.mysql.com/downloads/connector.odbc/`

The LabVIEW Database Connectivity Toolkit requires the 32-bit driver, but to define the data source name on Windows, the 64-bit divers are needed. When installing the dirvers, make sure to select the Complete Install option when prompted.

To set up the data source on Windows:

1. Go to Control Panel → Administrative Tools → Data Sources (ODBC)
2. Click Add
3. Select the MySQL ODBC ANSI Driver (Most recent version listed and click Finish
4. In the following pop up Window, enter the following values

   Data Source Name: Spinlab DB

   TCP/IP Server: `spinlabdb.nscl.msu.edu`

   User: `spinlab_app`

   Password: Provided per device upon request
5. Press Test to verify the connection
6. Click OK to complete setting up the data source

At this point, the `Spinlab DB.udl` file in the Database VI directory will properly connect to the SpinlabSC database, and the VI's will properly be able to read and write data.