openag_python Documentation

Release 0.1.5

Open Agriculture Initiative

Jun 07, 2017

Contents

This library is the core of OpenAg's backend software. The code and accompanying documentation define many of the standards on which the rest of the software is built. It defines the object model for the database, the structure of Arduino modules, and a command line interface for interacting setting up and interacting with a system.

There is also ROS package openag_brain which runs on the food computer. It is built on top of this library and provides things like control loops, data persistence, and taking images from a USB camera.

CHAPTER 1

Command Line Interface

This package provides a command line interface for performing all major functions required for setting up and managing a food computer instance.

Cloud

The subcommand `openag cloud` provides tools for selecting a cloud server to use, managing your user account on that server, and managing a farm instance on the server which serves as a mirror for your local instance.

```
Usage: openag cloud init [OPTIONS] CLOUD_URL

  Choose a cloud server to use. Sets CLOUD_URL as the cloud server to use
  and sets up replication of global databases from that cloud server if a
  local database is already initialized (via `openag db init`).

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud show [OPTIONS]

  Shows the URL of the current cloud server or throws an error if no cloud
  server is selected

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud deinit [OPTIONS]

  Detach from the current cloud server

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud register [OPTIONS]

  Create a new user account. Creates a user account with the given
  credentials on the selected cloud server.

Options:
  --username TEXT  Username for the account
  --password TEXT  Password for the account
  --help           Show this message and exit.
```

```
Usage: openag cloud login [OPTIONS]

  Log into your user account

Options:
  --username TEXT  Username for the account
  --password TEXT  Password for the account
  --help           Show this message and exit.
```

```
Usage: openag cloud show [OPTIONS]

  Shows the URL of the current cloud server or throws an error if no cloud
  server is selected

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud logout [OPTIONS]

  Log out of your user account

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud create_farm [OPTIONS] FARM_NAME

  Create a farm. Creates a farm named FARM_NAME on the currently selected
  cloud server. You can use the `openag cloud select_farm` command to start
  mirroring data into it.

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud list_farms [OPTIONS]

  List all farms you can manage. If you have selected a farm already, the
  name of that farm will be prefixed with an asterisk in the returned list.

Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud init_farm [OPTIONS] FARM_NAME

  Select a farm to use. This command sets up the replication between your
  local database and the selected cloud server if you have already
  initialized your local database with the `openag db init` command.
```

```
Options:
  --help  Show this message and exit.
```

```
Usage: openag cloud deinit_farm [OPTIONS]

  Detach from the current farm. Cancels the replication between your local
  server and the cloud instance if it is set up.

Options:
  --help  Show this message and exit.
```

DB

The subcommand `openag db` provides tools for managing your local CouchDB instance.

```
Usage: openag db init [OPTIONS]

  Initialize the database server. Sets some configuration parameters on the
  server, creates the necessary databases for this project, pushes design
  documents into those databases, and sets up replication with the cloud
  server if one has already been selected.

Options:
  --db_url TEXT
  --api_url TEXT
  --help          Show this message and exit.
```

```
Usage: openag db show [OPTIONS]

  Shows the URL of the current local server. Throws an error if no local
  server is selected

Options:
  --help  Show this message and exit.
```

```
Usage: openag db load_fixture [OPTIONS] FIXTURE_FILE

  Populate the database from a JSON file. Reads the JSON file FIXTURE_FILE
  and uses it to populate the database. Fuxture files should consist of a
  dictionary mapping database names to arrays of objects to store in those
  databases.

Options:
  --help  Show this message and exit.
```

```
Usage: openag db deinit [OPTIONS]

  Detach from the local server.

Options:
  --help  Show this message and exit.
```

```
Usage: openag db clear [OPTIONS]

  Clear all data on the local server. Useful for debugging purposed.

Options:
  --help  Show this message and exit.
```

Firmware

The subcommand `openag firmware` provides tools for generating and compiling code to run on the microcontroller of the system.

```
Usage: openag firmware init [OPTIONS]

  Initialize an OpenAg-based project

Options:
  -b, --board TEXT        The board to use for compilation. Defaults to
                          megaatmega2560 (Arduino Mega 2560)
  -d, --project-dir TEXT  The directory in which the project should reside
  --help                  Show this message and exit.
```

```
Usage: openag firmware run [OPTIONS]

  Generate code for this project and run it

Options:
  -d, --project-dir TEXT          The directory in which the project should
                                  reside
  --status_update_interval INTEGER
                                  Minimum interval between driver status
                                  updates (in seconds)
  -t, --target TEXT               PlatformIO target (e.g.  upload)
  -p, --plugin TEXT               Enable a specific plugin
  -f, --param_file FILENAME       YAML or JSON file describing the firmware
                                  module configuration to be flashed.
                                  This is
                                  the same file that is used for rosparam in
                                  the launch file.code
  -c, --categories [sensors|actuators|calibration|persistence|control]
                                  A list of the categories of inputs and
                                  outputs that should be enabled
  --help                          Show this message and exit.
```

```
Usage: openag firmware run_module [OPTIONS] [ARGUMENTS]...

  Run a single instance of this module. [ARGUMENTS] specifies a list of
  implementation-specific arguments to the module (for example, configuring
  Arduino pin numbers for the module).

  Example:

  openag firmware run_module -t upload 4

  This command fetches module definitions from CouchDB. CouchDB must be
```

```
  running on port 5984 and the firmware_module_type database populated with
  appropriate type records for this command to work. Loading the default
  fixture from openag_brain will populate a default set of
  firmware_module_type records.

Options:
  -b, --board TEXT                The board to use for compilation. Defaults
                                  to megaatmega2560 (Arduino Mega 2560)
  -d, --project-dir TEXT          The directory in which the project should
                                  reside
  --status_update_interval INTEGER
                                  Minimum interval between driver status
                                  updates (in seconds)
  -t, --target TEXT               PlatformIO target (e.g.  upload)
  -p, --plugin TEXT               Enable a specific plugin
  -f, --param_file FILENAME       YAML or JSON file describing the firmware
                                  module configuration to be flashed.
                                  This is
                                  the same file that is used for rosparam in
                                  the launch file.code
  -c, --categories [sensors|actuators|calibration|persistence|control]
                                  A list of the categories of inputs and
                                  outputs that should be enabled
  --help                          Show this message and exit.
```

CHAPTER 2

Object Model

openag.models.**Environment**

An `Environment` abstractly represents a single homogenous climate-controlled volume within a system. A food computer usually consists of a single `Environment`, but larger systems will often contain more than one `Environment`.

name

(str) A human-readable name for the environment

openag.models.**EnvironmentalDataPoint**

An *EnvironmentalDataPoint* represents a single measurement or event in an *Environment*, such as a single air temperature measurement or the start of a recipe.

environment

(str, required) The ID of the environment for which this point was measured

variable

(str, required) The type of measurement of event this represents (e.g. "air_temperature"). The class `EnvVar` contains all valid variable names.

is_manual

(bool) This should be true if the data point represents a manual reading performed by a user and false if it represents an automatic reading from a firmware or software module. Defaults to false.

is_desired

(bool, required) This should be true if the data point represents the desired state of the environment (e.g. the set points of a recipe) and false if it represents the measured state of the environment.

value

The value associated with the measurement or event. The exact use of this field may very depending on the *variable* field.

timestamp

(float, required) A UNIX timestamp reflecting when this data point was generated.

openag.models.**Recipe**

In order to allow for recipes to evolve, we have developed a very generic recipe model. The idea behind the model is that the system runs a recipe handler module which declares some list of recipe formats that it supports.

Recipes also declare what format they are. Thus, to define a new recipe format, you can write a custom recipe handler module type that understands that format, write recipes in the new format, and then use the rest of the existing system as is. See *Writing Recipes* for information on existing recipe formats and how to write recipes with them.

name
(str) A human-readable name for the recipe

description
(str) A description of the recipe and what it should be used for

format
(str, required) The format of the recipe

operations
(required) The actual content of the recipe, organized as specified for the format of this recipe

openag.models.**FirmwareInput**
A *FirmwareInput* gives information about a single input to a firmware module (a ROS topic to which the module subscribes). These objects are only ever stored in the *input* attribute of a *FirmwareModuleType* or *FirmwareModule*.

type
(str) The name of the ROS message type expected for messages on the topic

variable
(str) The name of the environmental variable affected by this input. For example, for a heater, this should be "air_temperature". Defaults to the key for this object in the parent dictionary.

categories
(list) A list of categories to which this inputs belongs. Must be a subset of ["actuators", "calibration"]

description
(str) A short description of what the input is for

multipler
(float) A factor by which to multiply data points on this input before they reach the module itself. This should generally be used to specify the extent to which the module affects the variable. For example, for an input which represents the command to send to a chiller module, the input should have the variable "air_temperature" and should have a negative multiplier so that a negative output from the air temperature control loop turns the chiller on. Fractional multipliers are allowed and can be useful to balance things from the perspective of the control loop when an up actuator (e.g. heater) is more powerful than its corresponding down actuator (e.g. chiller) or vice versa. Defaults to 1.

deadband
(float) Data points sent to this input with an absolute value less than the deadband will be sent as zeros instead. This is expecially useful for boolean inputs. For example, if a control loop outputs a float that is being fed into a binary actuator, a deadband can be put on the input to the actuator to effectively set a threshold on the commanded control effect above which the acuator will turn on.

openag.models.**FirmwareOutput**
A *FirmwareOutput* gives information about a single outputs from a firmware module (a ROS topic to which the module publishes). These objects are only ever stored in the *output* attribute of a *FirmwareModuleType* or *FirmwareModule*.

type
(str) The name of the ROS message type expected for messages on the topic

variable
(str) The name of the environmental variable represented by this output. Defaults to the key for this object in the parent dictionary.

categories
(list) A list of categories to which this output belongs. Must be a subset of [”sensors”, “calibration”]

description
(str) A short description of what the output is for

accuracy
(float) The maximum error for measurements on this output. Used to decide how to round the values before they are presented to the user.

repeatability
(float) A value below which the absolute difference between two repeated readings on this output should be expected to lie with a probability of 95% assuming that the underlying environmental condition is constant between readings.

openag.models.**FirmwareArgument**
A *FirmwareArgument* gives information about a single argument to a firmware module (an argument to the constructor for the Arduino class for the module). These objects are only ever stored in the *arguments* attribute of a *FirmwareModuleType* or *FirmwareModule*.

name
(str) The name of the argument

type
(str, required) Must be one of “int”, “float”, “bool”, and “str”

description
(str) A short description of what the argument is for

default
The value that should be used for the argument if the user doesn't specify one.

openag.models.**FirmwareModuleType**
A *FirmwareModuleType* represents a firmware library for interfacing with a particular system peripheral. It is essentially a driver for a sensor or actuator. The code can be either stored in a git repository or registered with PlatformIO and metadata about it should be stored in the OpenAg database. See *Writing Firmware Modules* for information on how to write firmware modules.

repository
(dict) A dictionary that describes where the code for this module type is hosted. The dictionary must always have the field “type” which indicates what service hosts the code. For a module hosted by platformio, this dictionary should have a “type” of “pio” and an “id” which is the integer ID of the platformIO library. For a module hosted in a git repository, the dictionary should have a “type” of “git” and a “url” which is the URL of the git repository.

header_file
(str, required) The name of the header file containing the top-level class in the library

class_name
(str, required) The name of the top-level class in the library

description
(str) Description of the library

categories
(list) A list of categories to which this firmware module type belongs. Must be a subset of [”sensors”, “actuators”, “calibration”].

arguments
(list) A list of *FirmwareArgument* objects representing the arguments to be passed to the constructor of the top-level class of this module. All arguments with a default value should be at the end of the list.

inputs
(dict) A nested dictionary mapping names of topics to which modules of this type subscribe to *FirmwareInput* objects describing those inputs.

outputs
(dict) A nested dictionary mapping names of topics to which modules of this type publish to *FirmwareOutput* objects describing those outputs.

dependencies
(dict) A list of libraries on which this module depends. In particular, it should be a list of dictionaries with the same structure as is required by the "repository" field.

status_codes
(dict) A dictionary mapping status codes (as 8-bit integers) for this module to strings describing the relevant status.

openag.models.**FirmwareModule**
A *FirmwareModule* is a single instance of a *FirmwareModuleType* usually configured to control a single physical sensor or actuator.

type
(str, required) The ID of the *FirmwareModuleType* of this object

environment
(str, required) The ID of the *Environment* on which this peripheral acts

categories
(list) A list of categories to which this firmware module belongs. Must be a subset of ["sensors", "actuators", "calibration"]. If a value for this attribute is provided, it will overwrite the value from the *FirmwareModuleType* for this module.

arguments
(list) A list of argument values to pass to the module. There should be at least as many items in this list as there are arguments in the *FirmwareModuleType* for this module that don't have a default value.

inputs
(dict) A nested dictionary mapping names of topics to which this module subscribes to `FirmwareInput` objects describing those inputs. The set of keys in this dictionary must be a subset of the keys in the *inputs* dictionary for the *FirmwareModuleType* for this module. Values in this dictionary override values in the firmware module type.

outputs
(dict) A nested dictionary mapping names of topics to which this module publishes to `FirmwareOutput` objects describing those outputs. The set of keys in this dictionary must be a subset of the keys in the *outputs* dictionary for the *FirmwareModuleType* for this module. Values in this dictionary override values in the firmware module type.

openag.models.**SoftwareModuleType**
A `SoftwareModuleType` is a ROS node that can be run on the controller for the farm (e.g. Raspberry Pi). It can listen to ROS topics, publish to ROS topics, and advertize services. Examples include the recipe handler and individual control loops. Software module types are distributed as ROS packages.

package
(str, required) The name of the ROS package containing the code for this object

executable
(str, required) The name of the executable for this object

description
(str) Description of the library

categories
(list) A list of categories to which this software module type belongs. Must be a subset of [”sensors”, “actuators”, “control”, “calibration”, “persistence”].

arguments
(array, required) An array of dictionaries describing the command line arguments to be passed to this module. The inner dictionaries must contain the field “name” (the name of the argument) and can contain the fields “type” (one of “int”, “float”, “bool”, and “str”), “description” (a short description of what the argument is for), “required” (a boolean indicating whether or not this argument is required to be passed to the module. defaults to False) and “default” (a default value for the argument in case no value is supplied). An argument should only have a default value if it is required.

parameters
(dict, required) A nested dictionary mapping names of ROS parameters read by this module to dictionaries describing those parameters. The inner dictionaries can contain the fields “type” (one of “int”, “float”, “bool”, and “str”) “description” (a short description of what the parameter is for), “required” (a boolean indicating whether or not this parameter is required to be defined), and “default” (a default value for the parameter in case no value is supplied). A parameter should only have a default value if it is required.

inputs
(dict) A nested dictionary mapping names of topics to which this library subscribes to dictionaries containing information about those topics. The inner dictionaries must contain the field “type” (the ROS message type expected for messages on the topic) and can contain the field “description” (a short description of what the input is for).

outputs
(dict) A nested dictionary mapping names of topics to which this library publishes to dictionaries containing information about those topics. The inner dictionary must contain the field “type” (the ROS message type expected for messages on the topic) and can contain the field “description” (a short description of what the output is for).

openag.models.**SoftwareModule**
A `SoftwareModule` is a single instance of a *SoftwareModuleType*.

type
(str, required) The ID of the *SoftwareModuleType* of this object

namespace
(str) The name of the ros namespace that should contain the ROS node for this software module. If no value is provided, the environment field is used instead. If no environment is provided, the module is placed in the global namespace.

environment
(str) The ID of the *Environment* on which this `SoftwareModule` acts.

categories
(list) A list of categories to which this software module belongs. Must be a subset of [”sensors”, “actuators”, “control”, “calibration”, “persistence”]. If a value for this attribute is provided, it will overwrite the value from the *SoftwareModuleType* for this module.

arguments
(array) A list of argument values to pass to the module. there should be at least as many items in this list as there are arguments in the *SoftwareModuleType* for this module that don’t have a default value.

parameters
(dict) A dictionary mapping ROS parameter names to parameter values. These parameters will be defined in the roslaunch XML file under the node for this software module.

mappings
(dict) A dictionary mapping ROS names for topics or parameters to different ROS names. Keys are the

names defined in the software module type and values are the names that should be used instead. This can be used, for example, to route the correct inputs into a control module with generic input names like *set_point* and *measured*.

CHAPTER 3

Database Names

The CouchDB server has a single database for each type of object defined in the object model.

Environment objects are stored in the "environment" database.

EnvironmentalDataPoint objects are stored in the "environmental_data_point" database.

Recipe objects are stored in the "recipes" database.

FirmwareModuleType objects are stored in the "firmware_module_type" database.

FirmwareModule objects are stored in the "firmware_module" database.

SoftwareModuleType objects are stored in the "software_module_type" database.

SoftwareModule objects are stored in the "software_module" database.

CHAPTER 4

Querying the Database

CouchDB has built in REST API that runs port 5984 that can be used to pull data from the database. For many of the databases used by this project, the built in API is sufficient because you usually want to retrieve all of the documents in the database. This can be done by using the _all_docs endpoint for the database in question. For example:

```
curl localhost:5984/<db_name>/_all_docs?include_docs=True
```

Environmental Data Points

For the *environmental_data_point* database, however, retreiving all of the documents is typically far too expensive because data is constantly being added to it. Because of this, the project defines a design document for this database with a couple of views and a list function that should prove useful.

By Timestamp

There is a *by_timestamp* view that sorts the data points by environment and then by timestamp. In particular, each data point gets mapped to a key of the format:

```
[<environment_id>, <timestamp>]
```

Querying the *by_timestamp* view is especially useful for getting all of the data points between a given time range for a specific environment. For example:

```
curl -g localhost:5984/environmental_data_point/_design/openag/_view/by_timestamp?
↪startkey=[%22environment_1%22,<start_timestamp>]\&endkey=[%22environment_1%22,<end_
↪timestamp>]
```

By Variable

There is also a *by_variable* view that sorts the data points by environment, then by whether they are measured or desired, then by variable, then by timestamp. In particular, each data point gets mapped to a key of the format:

```
[<environment_id>, "desired"/"measured", <variable>, <timestamp>]
```

It also has a reduce function which returns the data point with the largest timestamp.

The *by_variable* view can be used to get the most recent data point for each variable:

```
curl localhost:5984/environmental_data_point/_design/openag/_view/by_variable?group_
↪level=3
```

It can also be used to get the history of a particular variable over time:

```
curl -g localhost:5984/environmental_data_point/_design/openag/_view/by_variable?
↪reduce=false\&startkey=[%22environment_1%22,%22measured%22,<variable>]\&endkey=[
↪%22environment_1%22,%22measured%22,<variable>,{}]
```

CSV Dumps

There is a *csv* list function that can be used to output the results of a query to any of these views as a csv file. It takes a GET parameter *cols* which is a list of columns that should be included in the generated csv file. By default there are columns for "timestamp", "variable", and "value". For example, to output the history of a particular variable over time as a csv file with only the columns "timestamp" and "value":

```
curl -g localhost:5984/environmental_data_point/_design/openag/_list/csv/by_variable?
↪reduce=false\&startkey=[%22environment_1%22,%22measured%22,<variable>]\&endkey=[
↪%22environment_1%22,%22measured%22,<variable>,{}]\&cols=[%22timestamp%22,%22value
↪%22]
```

CHAPTER 5

Variable Types

The class `openag.var_types.EnvVar` contains a list of all of the environmental variables recognized by this system. This is a working list, so we will readily accept additions to it. The following is a copy of that list with descriptions of each variable

`openag.var_types.`**`AIR_TEMPERATURE`**
: Temperature of the air in degrees Celcius

`openag.var_types.`**`AIR_HUMIDITY`**
: A measure of the concentration of water in the air relative to the maximum concentration at the current temperature

`openag.var_types.`**`WATER_TEMPERATURE`**
: Temperature of the water in degrees Celcius

`openag.var_types.`**`WATER_POTENTIAL_HYDROGEN`**
: Potential Hydrogen of the water

`openag.var_types.`**`WATER_ELECTRICAL_CONDUCTIVITY`**
: Electrical conductivity of the water

`openag.var_types.`**`RECIPE_START`**
: Represents the start of a recipe

`openag.var_types.`**`RECIPE_END`**
: Represents the end of a recipe

`openag.var_types.`**`MARKER`**
: Marks some user-defined event

CHAPTER 6

Writing Recipes

There is currently only 1 supported recipe format, but the system is designed to allows new formats to be developed over time.

Simple Recipes

The "simple" recipe format conceptualizes recipes as a sequential list of set points for environmental variables. It doesn't take into account the expression of the plants being grown at all.

In particular, a "simple" recipe is a list of 3-element lists with the following structure:

```
[<offset>, <variable_type>, <value>]
```

Where <*offset*> is the number of seconds since the start of the recipe at which this set point should take effect, <*variable_type*> is the variable type to which the set point refers (e.g. "air_temperature"), and <*value*> is the value of the set point. The set point stays in effect until a new set point for that variable type is reached. The list of set points must be ordered by offset.

The recipe will end as soon as the last set point is emitted. Because of this, it is recommended to end the recipe with a *recipe_end* set point that indicates that the recipe should be stopped. The *value* field for that set point could be set to the empty string ("").

See this gist for an example of a recipe.

CHAPTER 7

Writing Firmware Modules

Overview

Firmware modules should be subclasses of the `Module` class defined in the OpenAg Firmware Module repository. They must define a `begin()` function that initializes the module itself. This `begin()` function will be called in the `setup()` function of the Arduino sketch generated for the project. They must also define an `update()` function that updates the module (e.g. reads from a sensor at some rate). This `update()` function will be called in the `loop()` function of the Arduino sketch generated for the project. The `Module` superclass defines a `status_level` attribute which the firmware module should use to report its current status. Valid value for this attribute (all defined in the header file for the `Module` superclass) are `OK` (which means that the module is "ok"), `WARN` (which means that there is some warning for the module), and `ERROR` (which means that there is an error preventing the module from working as desired). The superclass also defines a `status_msg` attribute which is a `String` that the firmware module should use to describe the status of the module. This is generally an empty string when the status level is "ok" and an error message when the status level is "warn" or "error". Finally, the superclass defines a `status_code` attribute which is a **:spp:class:'uint8_t'** value that the firmware module should use to describe the status of the module. This serves the same purpose as the `status_msg` field. The *module.json* file (described in more detail below) should contain a dictionary explaining the meaning of all valid `status_code` values for the module.

In addition to these standard functions and attributes (which are all defined in the header file for the `Module` class), the module must define a get function for each of its outputs and a set function for each of its inputs. In particular, it must define a get function of the following form for each output.

bool **`get_OUTPUT_NAME`**(OUTPUT_TYPE *&msg*)

The function takes as argument an object of the desired message type, populates the object with the current value of the output and returns True if and only if the message should be published on the module output.

The module must also define a set function of the following form for each input.

void **`set_INPUT_NAME`**(INPUT_TYPE *msg*)

The function takes as argument an object of the desired message type populated with the value being passed in as input and should immediately process the message.

In addition the module should define a *module.json* file containing all of the metadata about the firmware module. In particular, it should be an instance of the *`openag.models.FirmwareModuleType`* schema encoded as JSON.

The system uses PlatformIO to compile Arduino sketches, so modules must also define a *library.json* file meeting the PlatformIO specifications. To work with our system, this file need only contain the fields *name* and *framework*. The *name* field should be the name of the module, and the *framework* field should have the value *arduino*.

CHAPTER 8

Categories

The OpenAg system defines a list of "categories" which can be used to describe the functionality contained in a firmware/software module/input/output. For example, the firmware module for the am2315 sensor itself belongs to the "sensors" and "calibration" category because it outputs sensor data and has inputs for calibration. The air temperature and air humidity outputs from this firmware module belong to the "sensors" category because they represent sensor readings, and the inputs to this module used for calibration belong to the "calibration" category.

When flashing an Arduino, it is possible to specify a list of categories that should be enabled. By default, all categories are enabled except for "calibration". This allows the codegen system to generate one Arduino sketch to use during normal operation and a different sketch to use for calibration that enables the "calibration" inputs and disables the "actators", for example.

Examples

The repository openag_firmware_examples provides some examples of well-documented, simple firmware modules for reference.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index

W